IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC., Defendant.	Case No. 7:24-CV-00030 **JURY TRIAL DEMANDED**

DEFENDANTS' MOTION TO DISMISS

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Cases*:** ***Page(s):

Statutes and Rules:

I. INTRODUCTION

VirtaMove's complaint (1) fails to state any claim for relief against two of the three named defendants, (2) fails to state a claim for willful or indirect infringement against any defendant, and (3) fails to plead compliance with the patent marking statute, as Federal Circuit precedent requires.

First, VirtaMove accuses three separate defendants of infringing its patents, but the complaint's factual allegations pertain solely to products offered by one defendant—Amazon Web Services, Inc. ("AWS"). VirtaMove extends its allegations to the other two defendants by merely grouping all three defendants together under a single name ("Amazon" or "Defendant"). These grouped allegations fail to provide the requisite notice of what the two non-AWS defendants are accused of doing. Without factual allegations specific to the non-AWS defendants, VirtaMove fails to state any plausible claim against them.

Second, VirtaMove's complaint fails to state a plausible claim of willful or indirect infringement against any of the three defendants. The complaint merely recites the legal elements of these causes of action, without any factual allegations that might support them. Willful and indirect infringement each require knowledge of the patent-in-suit *and* knowledge that the patent is infringed. VirtaMove's complaint fails with respect to both knowledge elements. The complaint lacks any facts plausibly showing that the defendants knew of an asserted patent before this case, let alone that they knew of the alleged infringement. Without such factual allegations, the complaint's claims of willful and indirect infringement must be dismissed.

Third, VirtaMove fails to plead that it complied with the patent marking statute. That statute requires a patentee to provide notice to the public that its products practice a patent by marking such products with the patent number or an Internet address that identifies the patent number. Because VirtaMove fails to plead compliance, its claim for pre-suit damages must be dismissed.

II. ARGUMENT

A. Legal Standard

A complaint must contain "a short and plain statement of the claim showing that the pleader is entitled to relief[.]" Fed. R. Civ. P. 8(a)(2). A party may move to dismiss a complaint for "failure to state a claim upon which relief can be granted." Fed. R. Civ. P. 12(b)(6). To survive a motion to dismiss, the complaint "must contain sufficient factual matter, accepted as true, to 'state a claim to relief that is plausible on its face.'" *Ashcroft v. Iqbal,* 556 U.S. 662, 678, 129 S.Ct. 1937 (2009) (quoting *Bell Atl. Corp. v. Twombly*, 550 U.S. 544, 570 (2007)). "A claim has facial plausibility when the plaintiff pleads factual content that allows the court to draw the reasonable inference that the defendant is liable for the misconduct alleged." *Iqbal*, 556 U.S. at 678.

To state a plausible claim, a complaint must contain "more than labels and conclusions, and a formulaic recitation of the elements of a cause of action will not do." *Twombly*, 550 U.S. at 555. "'[N]aked assertions' devoid of 'further factual enhancement,'" and "[t]hreadbare recitals of the elements of a cause of action, supported by mere conclusory statements," are not entitled to the presumption of truth. *Iqbal*, 556 U.S. at 678 (quoting *Twombly*, 550 U.S. at 557); *see also R2 Invs. LDC v. Phillips*, 401 F.3d 638, 642 (5th Cir. 2005) (the court should not "strain to find inferences favorable to the plaintiffs" nor accept "conclusory allegations, unwarranted deductions, or legal conclusions").

In patent cases, issues that are unique to patent law are governed by Federal Circuit precedent. *Woods v. DeAngelo Marine Exhaust* , Inc., 692 F.3d 1272, 1279 (Fed. Cir. 2012). Procedural issues not unique to patent law, such as the plausibility standard under Rule 12(b)(6), are governed by precedent of the regional circuit. *Bascom Glob. Internet Servs., Inc. v. AT&T Mobility LLC*, 827 F.3d 1341, 1347 (Fed. Cir. 2016).

B. The Court Should Dismiss All Claims Against The Non-AWS Defendants.

1. The Complaint's Factual Allegations Pertain Solely to AWS.

VirtaMove's complaint identifies only two accused products: "*AWS* End-of-Support Migration Program ('EMP')" and "*AWS* Elastic Container Service ('ECS')." (Dkt. 1 at ¶¶ 14, 24 (emphasis added).) Both products are offered by defendant *AWS*—as their names indicate. Indeed, the complaint supports its allegations against these two products with citations to an AWS white paper. (*Id.*)[1] The complaint pleads no facts that would explain how the two other defendants (Amazon.com, Inc. and Amazon.com Services LLC) infringe with these AWS products. (*Id.*)

2. The Complaint Relies on Improper Group Pleading to Ensnare the Non-AWS Defendants.

VirtaMove extends its allegations against the accused AWS products to all three defendants by defining them collectively as "Defendant" or "Amazon." (Dkt. 1 at 1.) The complaint's infringement claims rely on allegations against the collectively defined "Defendant" without ever specifying which individual defendant allegedly performed which action. (*Id.* at ¶¶ 12-30.) Courts in the Fifth Circuit routinely reject such "group pleading" because it fails to provide notice to each defendant of what that defendant is alleged to have done. *See, e.g.*, *Taylor v. IBM*, 54 Fed. App'x 794 (5th Cir. 2002) (affirming dismissal under 12(b)(6) where "Appellants failed to allege specific acts of [copyright] infringement by each defendant, thereby failing to adhere to the requirements of Fed.R.Civ.P. 8(a)"); *Rowland v. Sw. Corr., LLC*, No. 420CV00847ALMCAN, 2021 WL 4206409, at *13 (E.D. Tex. Aug. 17, 2021)) ("Plaintiff cannot hide behind group pleading."), report and recommendation adopted, No. 4:20-CV-847, 2021 WL 4191433 (E.D. Tex. Sept. 15, 2021); *Springboards to Educ., Inc. v. Kipp Found.*, No. 3:16-CV-2436-G, 2017 WL 3917701, at

[1] The complaint's citations to the AWS white paper reference the following web address: https://d1.awsstatic.com/windows/AWS_EMP_WindowsServer_Whitepaper_V2.1.pdf.

*4 (N.D. Tex. Sept. 7, 2017) ("In order to comply with Rule 8(a), it is necessary that [plaintiff] attribute specific acts of [trademark] infringement to each defendant."); *Gurganus v. Furniss et al.*, No. 3:15-CV-03964-M, 2016 WL 3745684, at *5 (N.D. Tex. July 13, 2016) ("group pleading fails to meet the pleading requirements of Federal Rule of Civil Procedure 8").[2]

Group pleading may be permitted in one particularized circumstance—namely, where "it can be reasonably inferred that each and every allegation is made against each individual defendant." *ACQIS LLC v. Lenovo Grp. Ltd.*, No. 6:20-CV-00967-ADA, 2022 WL 2705269, at *3 (W.D. Tex. July 12, 2022). But where allegations do not apply equally to each defendant, group pleading is improper. *E.g.*, *BlackBerry Ltd. v. Nokia Corp.*, No. 17-CV-155-RGA, 2018 WL 1401330, at *2 (D. Del. Mar. 20, 2018) (dismissing direct infringement claims against three out of four entities that were accused as a group because allegations were specific to one of the grouped defendants).

Here, the Court cannot reasonably infer that each and every infringement allegation applies to each named defendant because VirtaMove's factual allegations are specific to AWS products only and mention no role or participation by other defendants in their creation or operation. (*Supra*, p. 3.) Thus, group pleading is not permitted here and the Court should dismiss the non-AWS defendants. *E.g.*, *Mod. Font Applications, LLC v. Peak Rest. Partners, LLC*, No. 2:19-CV-221 TS, 2019 WL 3781051, at *3 (D. Utah Aug. 12, 2019) (dismissing infringement claims against one defendant where plaintiffs' factual allegations were specific to a different defendant); *BlackBerry*, 2018 WL 1401330, at *2 (dismissing infringement claims against all but one of the grouped

[2] Courts around the country routinely dismiss complaints for failing to distinguish between defendants. *E.g.*, *PLS-Pac. Laser Sys. v. TLZ Inc.*, No. 06-CV-4585 RMW, 2007 WL 2022020, at *11 (N.D. Cal. July 9, 2007); *Best Medical Int'l, Inc. v. Accuray, Inc.*, No. 10-cv-1043, 2011 WL 860423, at *6 (W.D. Pa. Mar. 9, 2011); *Via Vadis, LLC v. Skype, Inc.*, No. 11-cv-507, 2012 WL 2789733, at *1 (D. Del. July 6, 2012); *Holmes v. Allstate Corp.*, No. 11-cv-1543, 2012 WL 627238, at *22, 25 (S.D.N.Y. Jan. 27, 2012), adopted by 2012 WL 626262 (S.D.N.Y. Feb. 27, 2012); *Medina v. Bauer*, No. 02-cv-8837, 2004 WL 136636, at *6 (S.D.N.Y. Jan. 27, 2004).

defendants); *see also Promos Techs., Inc. v. Samsung Elecs. Co.*, No. CV 18-307-RGA, 2018 WL 5630585, at *3 (D. Del. Oct. 31, 2018) (dismissing infringement claims against all grouped defendants); *North Star Innovations, Inc. v. Toshiba Corp.*, No. CV 16-115-LPS-CJB, 2016 WL 7107230, at *2 (D. Del. Dec. 6, 2016) (recommending dismissal of infringement claims against both grouped defendants); *Automated Transaction LLC v. New York Cmty. Bank*, No. 12-CV-3070 JS ARL, 2013 WL 992423, at *4 (E.D.N.Y. Mar. 13, 2013) (dismissing infringement claims against both grouped defendants).

3. The Complaint Alleges No Factual Basis For Treating The Three Corporate Defendants As One.

The complaint alleges that defendants Amazon.com Services LLC and AWS are both subsidiaries of defendant Amazon.com, Inc. (Dkt. 1 at ¶¶ 7-8.) The defendants' corporate relationship does not excuse VirtaMove's failure to adequately plead infringement against each defendant. "It is a general principle of corporate law deeply ingrained in our economic and legal systems that a parent corporation … is not liable for the acts of its subsidiaries." *United States v. Bestfoods*, 524 U.S. 51, 61 (1998). A court "must start from the general rule that the corporate entity should be recognized and upheld, unless specific, unusual circumstances call for an exception." *Manville Sales Corp. v. Paramount Sys., Inc.*, 917 F.2d 544, 552 (Fed. Cir. 1990).

Here, VirtaMove has not alleged any facts that would make Amazon.com, Inc. liable for the actions of its subsidiary, AWS, besides mere corporate ownership. *See BlackBerry*, 2018 WL 1401330, at *2 (dismissing patent infringement claims against parent company where a "complaint does not state facts supporting the existence of an agency relationship" and fails to state "facts that justify piercing the corporate veil"); *M2M Sols. LLC v. Telit Commc'ns PLC*, No. CV 14-1103-RGA, 2015 WL 4640400, at *3 (D. Del. Aug. 5, 2015) (similar); *Automated Transaction*, 2013 WL 992423, at *4 & n.6 (mere ownership and "opportunity to exercise control" are insufficient to

plausibly plead that a parent corporation is liable for a subsidiary's alleged infringement). And, even if VirtaMove could somehow attribute the alleged actions of AWS to its parent (Amazon.com, Inc.), this would not support any liability for Amazon.com Services LLC. The complaint alleges that Amazon.com Services LLC is a separate subsidiary—i.e., a ***sibling*** of AWS, not its parent. (Dkt. 1 at ¶ 8.) Thus, VirtaMove fails to plausibly allege vicarious liability for the non-AWS defendants.

4. The Complaint Fails to Plausibly Allege Joint Infringement.

The Federal Circuit has held that a "third party's actions [can be] attributed to [an] alleged infringer" when the "alleged infringer conditions participation in an activity or receipt of a benefit upon performance of a step or steps of a patented method and establishes the manner or timing of that performance." *Akamai Techs., Inc. v. Limelight Networks, Inc.*, 797 F.3d 1020, 1023 (Fed. Cir. 2015). In this scenario, "the third party's actions are attributed to the alleged infringer such that the alleged infringer becomes the single actor chargeable with direct infringement." *Id.* To the extent VirtaMove is attempting to plead joint infringement under this legal standard (*see* Dkt. 1 at ¶¶ 15, 25), its allegations fail for two reasons.

First, VirtaMove's joint-infringement allegations are wholly conclusory. The complaint recites the legal elements of "conditioning benefits" and "establishing the timing and manner of infringement," with no supporting facts. (*Id.*) Such "'naked assertion[s]' devoid of 'further factual enhancement'" fail to state a claim under Rule 12(b)(6). *Iqbal*, 556 U.S. at 678.

Second, VirtaMove fails to allege that any single defendant is responsible for controlling all steps of a claimed method. (*Id.*) Instead, VirtaMove alleges that "***Defendant***"—defined collectively—"***and/or users*** of the Accused Products directs and controls [the infringing activity]." (Dkt. 1 at ¶¶ 15, 25 (emphasis added).) Because "Defendant" refers to three different defendants, and the "users" are not even named in this case, VirtaMove fails to allege that all steps of a claimed

method are controlled by a single defendant. This failure is fatal to VirtaMove's claim because infringement under the proffered legal theory requires that "all steps of a claimed method are performed by or attributable to ***a single entity***." *Akamai*, 797 F.3d at 1022.

Because VirtaMove's complaint does not allege any plausible basis to hold the non-AWS defendants liable for alleged infringement by the AWS products, the Court should dismiss the non-AWS defendants from this case.

C. The Court Should Dismiss All Claims That Require Pre-Suit Knowledge.

1. The Complaint Fails to Plausibly Allege Willful Infringement.

A plaintiff alleging willful patent infringement must "allege facts plausibly showing that the accused infringer: '(1) knew of the patent-in-suit; (2) after acquiring that knowledge, it infringed the patent; and (3) in doing so, it knew, or should have known, that its conduct amounted to infringement of the patent.'" *BillJCo*, *v. Apple Inc.,* 583 F. Supp. 3d 769, 774 (W.D. Tex. 2022) (internal quotations omitted). Importantly, "[m]ere knowledge of the Asserted Patents is not enough" to establish knowledge of infringement. *Id.* at 777; *CTD Networks, LLC v. Google, LLC*, No. WA-22-CV-01042-XR, 2023 WL 5417139, *9 (W.D. Tex. Aug. 22, 2023).

Here, VirtaMove alleges that all three defendants willfully infringe the asserted '814 patent. (Dkt. 1 at ¶ 16.) But VirtaMove alleges no facts plausibly showing that any of the three defendants (1) knew of the '814 patent or (2) knew or should have known that they were infringing the '814 patent.

VirtaMove's allegations of willful infringement are set forth in a single paragraph where it alleges that "Defendant" (defined collectively) "knew of VirtaMove, its products, and at least one of the patents long before this suit was filed and at least as early as 2013." (*Id.*) VirtaMove alleges two facts: (a) "on or about October 2013, in connection with the prosecution of U.S. Patent No. 8,806,655 (assigned to Amazon), the examiner cited the '814 Patent against Amazon"; and (b)

"On or about June 2020, in connection with the prosecution of U.S. Patent No. 11,061,812 (assigned to Amazon), the examiner cited U.S. Pub. No. 2005/0060722 (which issued as the '814 Patent) against Amazon." (*Id.*) VirtaMove then concludes that "Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent." (*Id.*) These allegations fail to state a plausible claim for willful infringement for two reasons.

a. VirtaMove Does Not Plausibly Allege that Any Defendant Had Knowledge of the Patent-in-Suit.

Plaintiff's complaint defines three separate entities as "Amazon" or "Defendant," and alleges that "Defendant" knew of the '814 patent (or its application) because it was cited during prosecution of an application "assigned to Amazon." (Dkt. 1 at ¶ 16.) But this allegation is not plausible because the patents on which the allegation relies identify their assignee as "Amazon Technologies, Inc." ("ATI"). (Ex. 1 at 1; Ex. 2 at 1.) ATI is not a defendant in this case.

The ATI patents are properly considered on this motion to dismiss because they are subject to judicial notice. *Tellabs, Inc. v. Makor Issues & Rts., Ltd.*, 551 U.S. 308, 322, 127 S. Ct. 2499, 2509 (2007) ("courts must consider … matters of which a court may take judicial notice" when ruling on a motion to dismiss); *Kaempe v. Myers*, 367 F.3d 958, 965 (D.C. Cir. 2004) (Patent Office filings "are public records subject to judicial notice on a motion to dismiss"). Additionally, the ATI patents are properly considered on this motion to dismiss because they are "referred to in the plaintiff's complaint and are central to [the plaintiff's] claim." *Collins v. Morgan Stanley Dean Witter*, 224 F.3d 496, 498–99 (5th Cir. 2000).

VirtaMove never alleges any facts that would allow the Court to impute any alleged knowledge from ATI to any defendant. Without some factual basis to impute such knowledge from a third-party (even a related third-party) to the named defendants, VirtaMove's allegations fail to state a claim of willfulness. *See, e.g.*, *Xiros, Ltd. v. Depuy Synthes Sales, Inc.*, No. W-21-

CV-00681-ADA, 2022 WL 3592449, at *3 (W.D. Tex. Aug. 22, 2022) (granting motion to dismiss willful infringement claim because knowledge could not be imputed from an affiliated company to the defendant); *Flypsi, Inc. v. Google LLC*, No. 6:22-CV-0031-ADA, 2022 WL 3593053, at *5 (W.D. Tex. Aug. 22, 2022) ("the Court will not entertain inferential connections between two companies without proper allegations of how the knowledge would flow from one to another"); *Core Optical Techs., LLC v. Nokia Corp.*, No. SACV1902190JAKRAOX, 2020 WL 6126285, at *7 (C.D. Cal. Oct. 8, 2020) (pre-suit knowledge was not plausible where the plaintiff's patent was cited as prior art against patents that defendant allegedly owned "indirectly"); *Dynamic Data Techs. v. Google LLC*, No. 19-1529, 2020 WL 128582, *3 (D. Del. March 18, 2020) (recommending granting a motion to dismiss a willful infringement claim against a subsidiary when allegations of knowledge related solely to the parent corporation); *ZitoVault, LLC v. Int'l Bus. Machines Corp.*, No. 3:16-CV-0962-M, 2018 WL 2971131, at *3 (N.D. Tex. Mar. 29, 2018) ("Plaintiff needs to set out, in its Complaint, more than just the bare facts of the parent/subsidiary relationship in order to plausibly allege that [the defendant] had knowledge of the [asserted] patent."); *Varian Med. Sys., Inc. v. Elekta AB*, No. 15-871-LPS, 2016 WL 3748772, at *5 (D. Del. July 12, 2016), report and recommendation adopted, 2016 WL 9307500 (D. Del. Dec. 22, 2016) (refusing to attribute knowledge of one corporate entity to another despite the affiliated entities sharing a trade-name); *Chalumeau Power Sys. LLC v. Alcatel-Lucent*, No. CIV.A. 11-1175-RGA, 2012 WL 6968938, at *1 (D. Del. July 18, 2012) ("The connection between the moving defendants and the patent applications from nearly a decade ago are not explained sufficiently to make plausible that either of the defendants had actual knowledge.").

Because VirtaMove pleads no facts plausibly showing that any of the three defendants—as opposed to ATI—knew of the asserted patent's existence, the complaint fails to state a claim for willful infringement.

b. Plaintiff Does Not Plausibly Allege Any Knowledge of Infringement.

Even if VirtaMove could somehow show that some defendant knew of the '814 patent, the willfulness claim would still fail because VirtaMove never plausibly alleges that each defendant knew that the accused AWS product infringed the patent. *CTD Networks, LLC v. Amazon.com, Inc.*, No. W-22-cv-01034-XR, 2023 WL 5281943, *7 (W.D. Tex. Aug. 16, 2023), *Monolithic Power Sys., Inc. v. Meraki Integrated Circuit (Shenzhen) Tech., Ltd.*, No. 6:20-cv-008876-ADA, 2021 WL 3931910, *5 (W.D. Tex. Sept. 1, 2021), *Dali Wireless Inc. v Corning Optical Communications LLC*, 638 F. Supp. 3d 1088, 1098-1100 (N.D. Cal. 2022). VirtaMove offers only the bare assertion that "Defendant knew, or should have known, that its conduct amounted to infringement of the '814 patent." (Dkt. 1 at ¶ 16.) But merely pleading an element of a cause of action is insufficient. *Twombly*, 550 U.S. at 555.

VirtaMove pleads no facts suggesting that anyone with knowledge of the '814 patent—whether at ATI or at one of the defendants—also knew how the allegedly infringing AWS service (EMP) operated, let alone understood that EMP was infringing the patent. *See Hills Point Indus. LLC v. Just Fur Love LLC*, No. CV 22-1256 (GBW), 2023 WL 8804046, *3-4 (D. Del. Dec. 20, 2023) (patent examiner's citation to asserted patent during prosecution by accused infringer was insufficient to support willfulness allegation) (citing *Callwave Commc'ns LLC v. AT & T Mobility LLC,* C.A. No. CV 12-1701-RGA, 2014 WL 5363741 at *2 (D. Del. Jan. 28, 2014)).

VirtaMove never alleges that EMP even existed in 2013—when the three defendant companies supposedly learned of the patent. (*See* Dkt. 1 at ¶ 16.) In fact, EMP did not launch until

December 2019.[3] Neither ATI nor any of the defendants could have plausibly known that EMP was allegedly infringing years before it existed.[4]

If VirtaMove wanted to make Amazon aware of any alleged infringement, VirtaMove could have sent a demand letter after EMP launched in 2019. But far from alleging infringement, VirtaMove has instead claimed to be "partner[ing]" with AWS since 2021.[5] All the while, VirtaMove never claims to have said anything to AWS about its patents. Thus, VirtaMove's willfulness allegations are meritless.

Because VirtaMove never plausibly alleges that any defendant knew of the '814 patent *and knew or should have known that the accused EMP product infringed that patent*, the complaint fails to state a claim of willful infringement.

2. The Complaint Fails to Plausibly Allege Indirect Infringement.

Both forms of indirect infringement (induced and contributory) require knowledge of the patent-in-suit *and* knowledge of infringement. *Global-Tech Appliances, Inc. v. SEB S.A.*, 563 U.S. 754, 766 (2011) (induced infringement requires "knowledge that the induced acts constitute patent

[3] Press release, Amazon Web Services, *AWS launches new program to drive migrations for end of support Windows Server applications* (Dec. 1, 2019), archived at: https://web.archive.org/web/20201031015900/aws.amazon.com/about-aws/whats-new/2019/12/aws-launches-program-drive-migration-windows-server/.

[4] VirtaMove's only allegation of knowledge after EMP's 2019 launch relates to a published patent *application* that was cited during ATI's prosecution in 2020. (Dkt. 1 at ¶ 16.) Notice of a mere application does not raise a plausible inference that ATI knew of the issued patent—much less the alleged infringement. *Meetrix IP, LLC v. Cisco Sys., Inc.*, No. 1-18-CV-309-LY, 2018 WL 8261315, at *2 (W.D. Tex. Nov. 30, 2018) (alleged notice of patent application does not plausibly show knowledge of issued patent) (citing *Vasudevan Software, Inc. TIBCO Software Inc.*, No. C 11-06638 RS, 2012 WL 1831543, at *3 (N.D. Cal. May 18, 2012)).

[5] Press release, VirtaMove, *VirtaMove Partners with AWS, Launches SaaS Product in AWS Marketplace* (April 28, 2021), https://virtamove.com/blog/virtamove-partners-with-aws-launches-saas-in-aws-marketplace/.

infringement"); 35 U.S.C. §271(c) (contributory infringement requires "knowing [that a component is] especially made or especially adapted for use in an infringement"). As shown above, VirtaMove failed to plausibly allege that any defendant had knowledge of the asserted patents, let alone that any defendant had knowledge of any infringement, before the complaint was filed. Thus, VirtaMove's complaint fails to state a claim for pre-suit indirect infringement.

D. The Court Should Dismiss VirtaMove's Claim for Pre-Suit Damages.

"Pursuant to 35 U.S.C. § 287(a), a patentee who makes or sells a patented article must mark his articles or notify infringers of his patent in order to recover damages." *Arctic Cat Inc. v. Bombardier Recreational Prod. Inc.*, 876 F.3d 1350, 1365 (Fed. Cir. 2017). "The patentee bears the burden of pleading and proving he complied with § 287(a)'s marking requirement." *Id.* at 1366. "At the motion to dismiss stage, '[a] claim for past damages requires pleading compliance with the marking statute[.]'" *CPC Pat. Techs. Pty Ltd. v. Apple Inc.*, No. 6:21-CV-00165-ADA, 2022 WL 118955, at *2 (W.D. Tex. Jan. 12, 2022) (citing *Express Mobile, Inc. v. DreamHost LLC*, No. 1:18-CV-01173-RGA, 2019 WL 2514418, at *2 (D. Del. June 18, 2019)).

VirtaMove's complaint fails to plead compliance with the marking statute. (*See generally* Dkt. 1.) The complaint never mentions marking or § 287.[6] Thus, the Court should dismiss VirtaMove's claims to the extent they seek past damages (i.e., damages for infringement that allegedly occurred before VirtaMove filed this case). *Express Mobile*, 2019 WL 2514418 at *2.

A patentee's failure to comply with the marking requirement may be excused if "the infringer was notified of the infringement." 35 U.S.C. § 287(a); *see CPC*, 2022 WL 118955, at *2. But such notice of infringement must come from the patentee—not a third party. *Lans v. Digital*

[6] For example, VirtaMove was required to mark at least its "V-Migrate" product because VirtaMove claims that the product practices both asserted patents. *See* VirtaMove, *Product Patents*, https://virtamove.com/about-us/product-patents/ (accessed April 3, 2024).

Equip. Corp., 252 F.3d 1320, 1328 (Fed. Cir. 2001). Here, the complaint never alleges that VirtaMove notified any defendant of the alleged infringement. For the '814 patent, the only notice that VirtaMove alleges is notice of the patent's existence (not the infringement) from the Patent Office (not VirtaMove) to ATI (not the defendants). And for the '058 patent, the complaint does not allege any notice at all. Thus, VirtaMove fails to plead compliance with § 287 and its claim for past damages must therefore be dismissed. *Express Mobile*, 2019 WL 2514418 at *2.

III. CONCLUSION

For the foregoing reasons, the Court should dismiss all claims against the non-AWS defendants. The Court should also dismiss the claims against AWS for willful infringement, indirect infringement, and damages, to the extent such claims encompass pre-suit conduct.

April 5, 2024

Respectfully submitted,

LYNCH, CHAPPELL & ALSUP
A Professional Corporation
Suite 700
300 N. Marienfeld,
Midland, Texas 79701
Telephone: 432-683-3351
Telecopier: 432-683-2587

By: */s/ Harper Estes*
Harper Estes
Texas Bar No. 00000083
hestes@lcalawfirm.com

Counsel for Defendants Amazon.com, Inc., Amazon.com Services, LLC and Amazon Web Services, Inc.

Of Counsel:
Colin B. Heidman *(Pro Hac Vice)*
Christie R.W. Matthaei *(Pro Hac Vice)*
Logan P. Young *(Pro Hac Vice)*
KNOBBE MARTENS OLSON & BEAR LLP
925 4th Ave, Ste 2500
Seattle, WA 98104
Telephone: 206-405-2000
Facsimile: 206-405-2001
colin.heideman@knobbe.com
christie.matthaei@knobbe.com
logan.young@knobbe.com

Joseph R. Re *(Pro Hac Vice)*
Jeremy A. Anapol *(Pro Hac Vice)*
KNOBBE MARTENS OLSON & BEAR LLP
2040 Main Street, 14th Floor
Irvine, CA 92614
Telephone: 949-760-0404
Facsimile: 949-760-9502
joe.re@knobbe.com
jeremy.anapol@knobbe.com

CERTIFICATE OF SERVICE

I hereby certify that on April 5, 2024, all counsel of record who are deemed to have consented to electronic service were served with a copy of the foregoing via the Court's CM/ECF System.

/s/ Harper Estes
Harper Estes

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF TEXAS
MIDLAND/ODESSA DIVISION

VIRTAMOVE, CORP., Plaintiff, v. AMAZON.COM, INC.; AMAZON.COM SERVICES LLC; AND AMAZON WEB SERVICES, INC., Defendant.	Case No. 7:24-CV-00030 **JURY TRIAL DEMANDED**

DECLARATION OF JEREMY A. ANAPOL
IN SUPPORT OF DEFENDANTS' MOTION TO DISMISS

I, Jeremy A. Anapol, declare and state as follows:

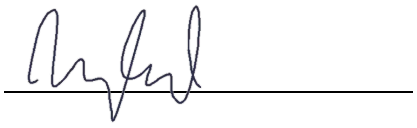
1. As counsel for Defendants Amazon.com, Inc., Amazon.com Services LLC, and Amazon Web Services, Inc., I will be seeking admission *pro hac vice* in the above-captioned case. I have personal knowledge of the matters set forth herein unless otherwise noted. If called upon to testify, I would testify competently thereto.

2. Attached hereto as Exhibit 1 is a true and correct copy of United States Patent No. 8,806,655, with highlighting added by me on the first page. This patent is referenced in VirtaMove's complaint. (Dkt. 1 at ¶ 16.)

3. Attached hereto as Exhibit 2 is a true and correct copy of United States Patent No. 11,061,812, with highlighting added by me on the first page. This patent is also referenced in VirtaMove's complaint. (Dkt. 1 at ¶ 16.)

I declare under penalty of perjury that the foregoing is true and correct.

Executed on April 5, 2024 at Irvine, California.



Jeremy A. Anapol

Exhibit 1

US008806655B1

(12) **United States Patent**
Brownell et al.

(10) **Patent No.: US 8,806,655 B1**
(45) **Date of Patent: Aug. 12, 2014**

(54) **PROVIDING LIMITED VERSIONS OF APPLICATIONS**

(75) Inventors: **David M. Brownell**, Renton, WA (US); **Gerard J. Heinz, II**, Seattle, WA (US); **Patrick G. McCuller**, Seattle, WA (US)

(73) Assignee: **Amazon Technologies, Inc.**, Reno, NV (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 189 days.

(21) Appl. No.: **13/288,320**

(22) Filed: **Nov. 3, 2011**

(51) **Int. Cl.**
G06F 7/04 (2006.01)

(52) **U.S. Cl.**
USPC **726/27**; 717/104; 717/178; 713/167; 709/214; 709/248; 707/707; 707/723; 707/731; 705/26.1

(58) **Field of Classification Search**
USPC 726/27; 717/104, 178; 713/167; 709/214, 248; 707/707, 723, 731; 705/26.1
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,519,814	B2 *	4/2009	Rochette et al.	713/167
2008/0133414	A1 *	6/2008	Qin et al.	705/50
2011/0066999	A1 *	3/2011	Rabinovich et al.	717/104

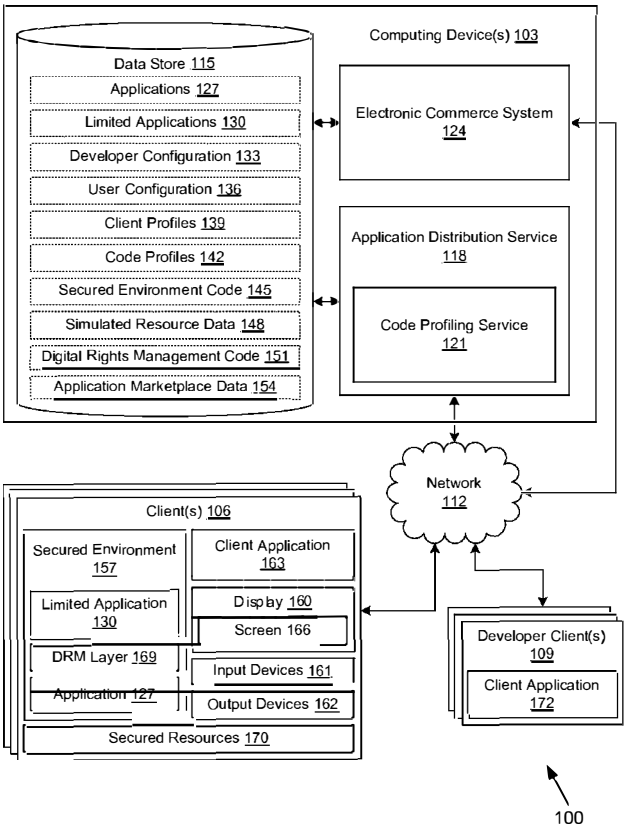
* cited by examiner

Primary Examiner — Thanhnga B Truong
(74) *Attorney, Agent, or Firm* — Thomas | Horstemeyer, LLP

(57) **ABSTRACT**

Disclosed are various embodiments for providing limited versions of applications. A limited version of an application is automatically generated from a full version of the application based at least in part on an expected use of the application by a client computing device during a testing period. The limited version has a smaller data size than the full version. The limited version of the application is sent to the client computing device. The limited version of the application is configured to be executed in a secured environment of the client computing device. The secured environment denies the limited version of the application access to secured resources of the client computing device.

24 Claims, 5 Drawing Sheets



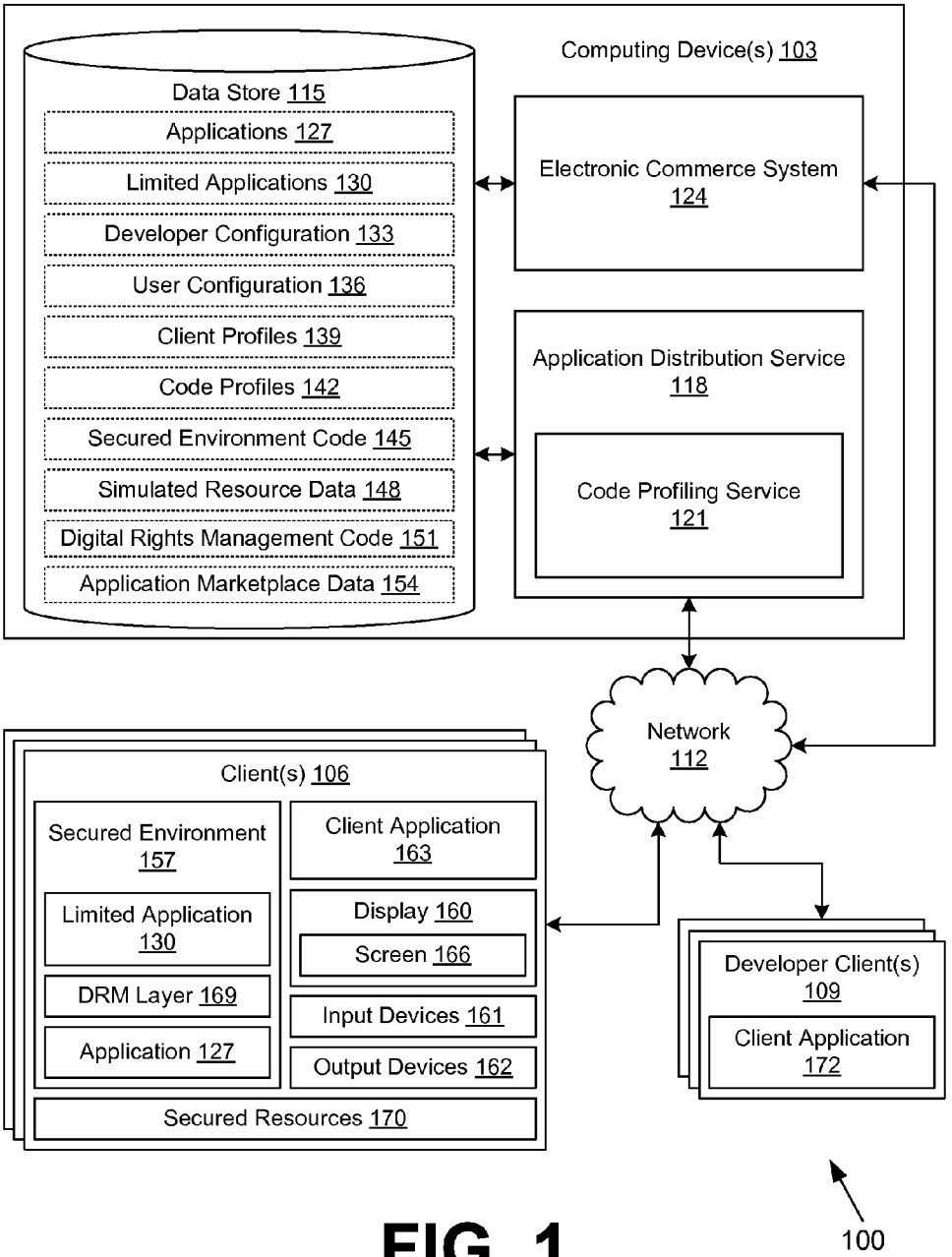

Computing Device(s) 103
Data Store 115
Applications 127
Limited Applications 130
Developer Configuration 133
User Configuration 136
Client Profiles 139
Code Profiles 142
Secured Environment Code 145
Simulated Resource Data 148
Digital Rights Management Code 151
Application Marketplace Data 154
Electronic Commerce System 124
Application Distribution Service 118
Code Profiling Service 121
Network 112
Client(s) 106
Secured Environment 157
Limited Application 130
DRM Layer 169
Application 127
Client Application 163
Display 160
Screen 166
Input Devices 161
Output Devices 162
Secured Resources 170
Developer Client(s) 109
Client Application 172
100

FIG. 1

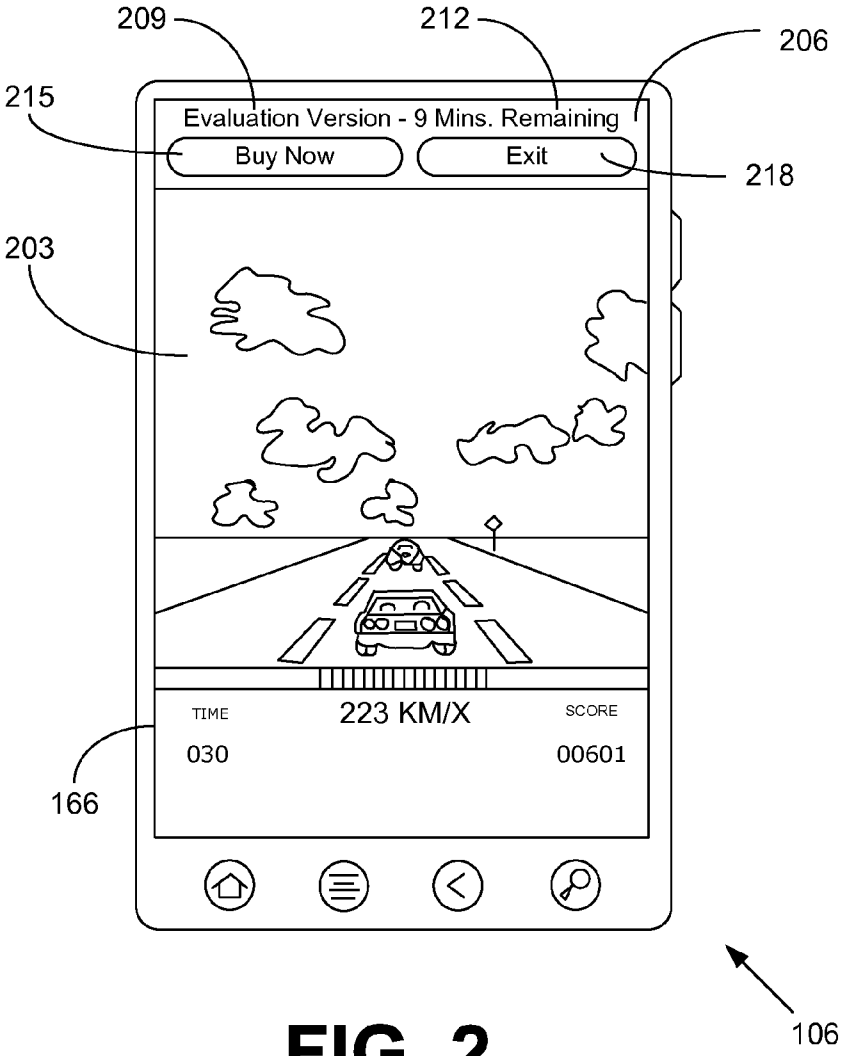

209
212
206
215
Evaluation Version - 9 Mins. Remaining
Buy Now
Exit
218
203
TIME
223 KM/X
SCORE
030
00601
166
106

FIG. 2

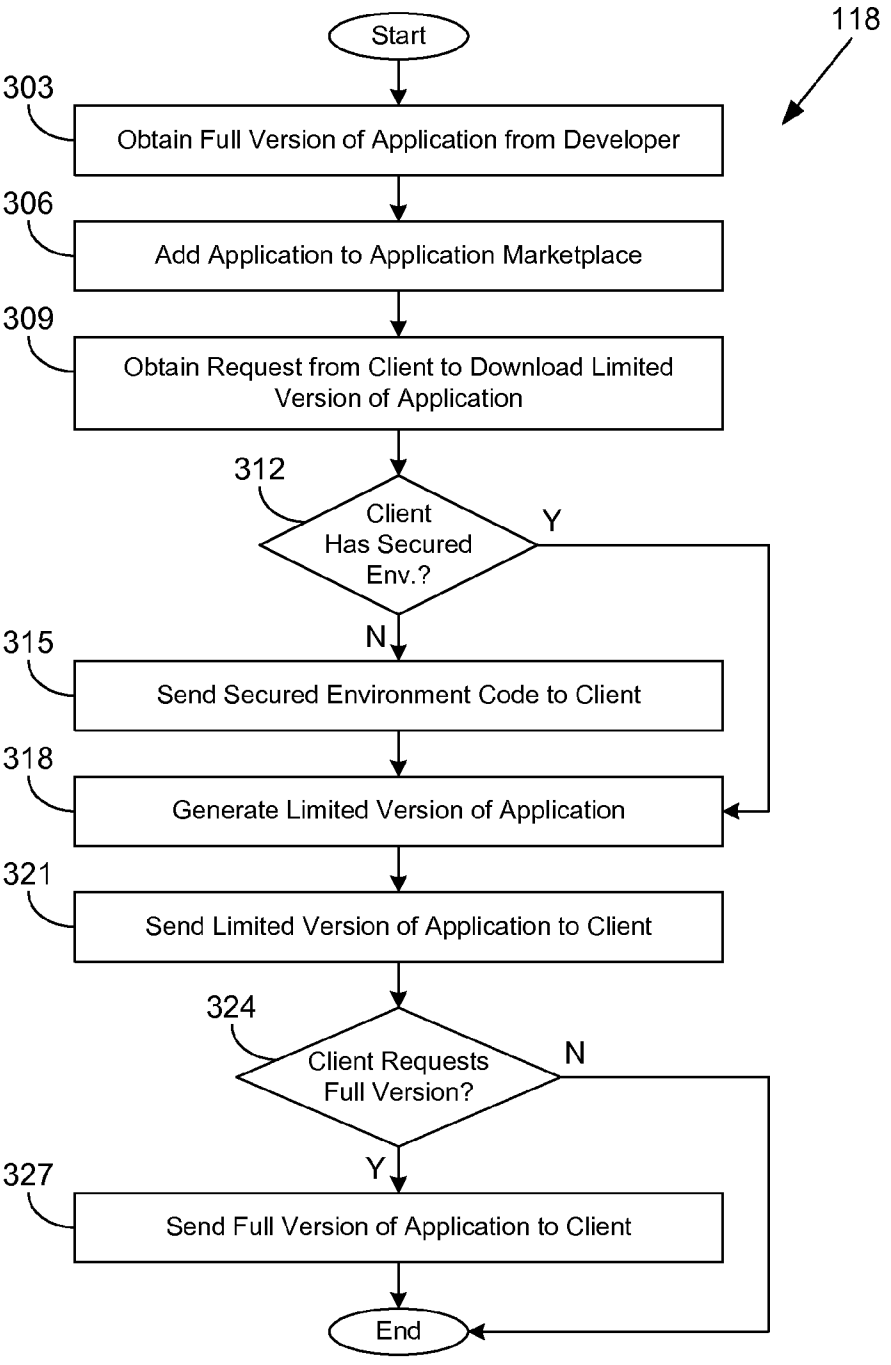

118
Start
303
Obtain Full Version of Application from Developer
306
Add Application to Application Marketplace
309
Obtain Request from Client to Download Limited Version of Application
312
Client Has Secured Env.?
Y
N
315
Send Secured Environment Code to Client
318
Generate Limited Version of Application
321
Send Limited Version of Application to Client
324
Client Requests Full Version?
N
Y
327
Send Full Version of Application to Client
End

FIG. 3

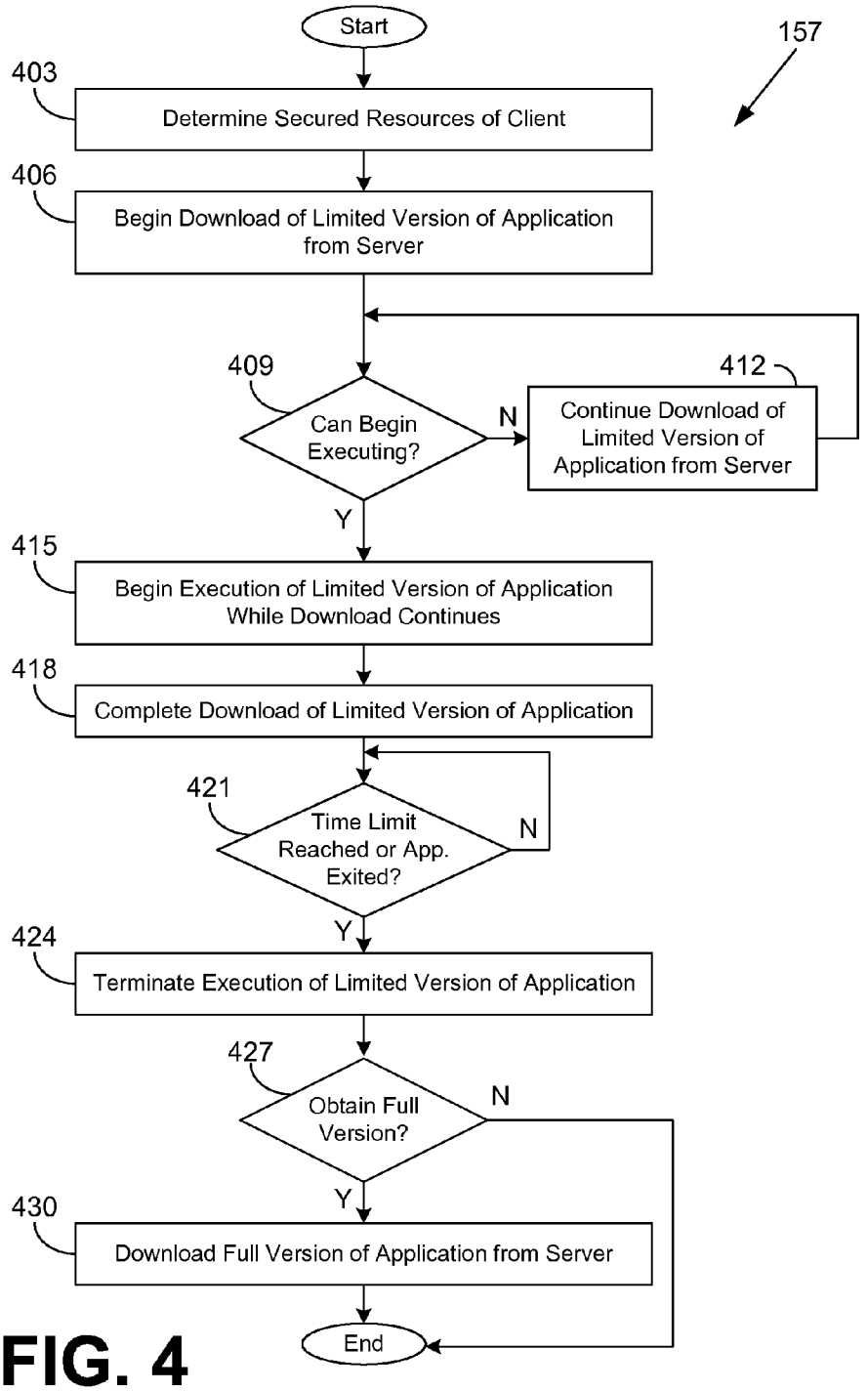

Start
157
403
Determine Secured Resources of Client
406
Begin Download of Limited Version of Application from Server
409
Can Begin Executing?
N
412
Continue Download of Limited Version of Application from Server
Y
415
Begin Execution of Limited Version of Application While Download Continues
418
Complete Download of Limited Version of Application
421
Time Limit Reached or App. Exited?
N
Y
424
Terminate Execution of Limited Version of Application
427
Obtain Full Version?
N
Y
430
Download Full Version of Application from Server
End

FIG. 4

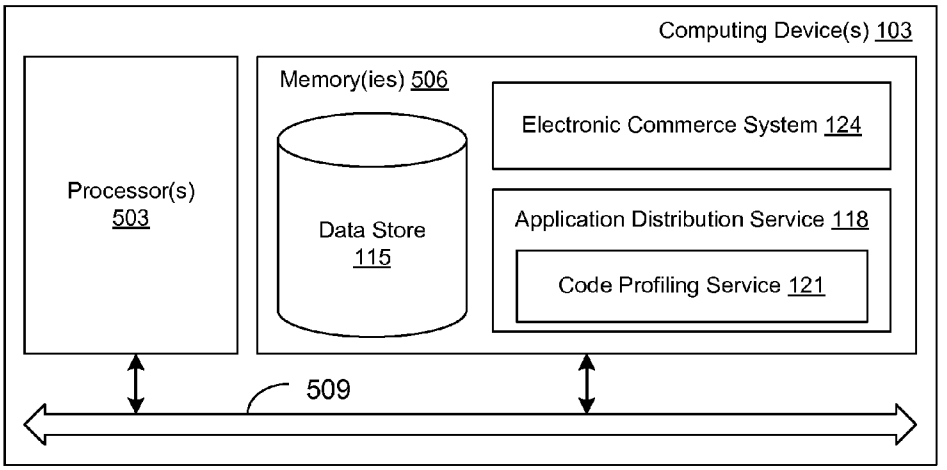

Computing Device(s) 103
Processor(s) 503
Memory(ies) 506
Data Store 115
Electronic Commerce System 124
Application Distribution Service 118
Code Profiling Service 121
509

FIG. 5

PROVIDING LIMITED VERSIONS OF APPLICATIONS

BACKGROUND

Software download via the Internet is quickly replacing traditional brick-and-mortar avenues for software distribution. Broadband Internet access allows a customer to download software that would otherwise be distributed through multiple compact discs (CDs) or digital versatile discs (DVDs). Such downloads may be more convenient than a trip to a retail store. Depending on the program size and connection speed, a program may be downloaded, for example, in seconds, minutes, or hours.

BRIEF DESCRIPTION OF THE DRAWINGS

Many aspects of the present disclosure can be better understood with reference to the following drawings. The components in the drawings are not necessarily to scale, emphasis instead being placed upon clearly illustrating the principles of the disclosure. Moreover, in the drawings, like reference numerals designate corresponding parts throughout the several views.

FIG. **1** is a drawing of a networked environment according to various embodiments of the present disclosure.

FIG. **2** is a drawing of an example of a client in the networked environment of FIG. **1** according to various embodiments of the present disclosure.

FIG. **3** is a flowchart illustrating one example of functionality implemented as portions of an application distribution service executed in a computing device in the networked environment of FIG. **1** according to various embodiments of the present disclosure.

FIG. **4** is a flowchart illustrating one example of functionality implemented as portions of a secured environment executed in a client in the networked environment of FIG. **1** according to various embodiments of the present disclosure.

FIG. **5** is a schematic block diagram that provides one example illustration of a computing device employed in the networked environment of FIG. **1** according to various embodiments of the present disclosure.

DETAILED DESCRIPTION

The present disclosure relates to providing limited versions of applications for trial and testing purposes. Mobile computing devices and/or other types of computing devices may have limited connection bandwidth. While Wi-Fi speeds may be acceptable for downloading full versions of applications, it may be that third generation (3G) or fourth generation (4G) bitrates are not. This is especially pertinent in the context of application trials. A user may be seeking to evaluate an application for purchase, but may not have the attention span to be willing to wait minutes for the application package to download. Also, users may be reluctant to install trial software on their computing devices due to security concerns.

Various embodiments of the present disclosure facilitate the generation of limited versions of applications for trial and testing purposes. The limited versions of applications may be distributed by progressive download and/or intelligent download to create a near-instantaneous testing experience for users upon commencing the download process. Furthermore, a secured environment, or "sandbox," may be provided in the device of the user to control access of the application to device resources. Digital rights management (DRM) functionality may be integrated into this environment to secure the downloaded code from unauthorized use. In the following discussion, a general description of the system and its components is provided, followed by a discussion of the operation of the same.

With reference to FIG. **1**, shown is a networked environment **100** according to various embodiments. The networked environment **100** includes one or more computing devices **103** in data communication with one or more clients **106** and one or more developer clients **109** by way of a network **112**. The network **112** includes, for example, the Internet, intranets, extranets, wide area networks (WANs), local area networks (LANs), wired networks, wireless networks, or other suitable networks, etc., or any combination of two or more such networks.

The computing device **103** may comprise, for example, a server computer or any other system providing computing capability. Alternatively, a plurality of computing devices **103** may be employed that are arranged, for example, in one or more server banks or computer banks or other arrangements. For example, a plurality of computing devices **103** together may comprise a cloud computing resource, a grid computing resource, and/or any other distributed computing arrangement. Such computing devices **103** may be located in a single installation or may be distributed among many different geographical locations. For purposes of convenience, the computing device **103** is referred to herein in the singular. Even though the computing device **103** is referred to in the singular, it is understood that a plurality of computing devices **103** may be employed in the various arrangements as described above.

Various applications and/or other functionality may be executed in the computing device **103** according to various embodiments. Also, various data is stored in a data store **115** that is accessible to the computing device **103**. The data store **115** may be representative of a plurality of data stores **115** as can be appreciated. The data stored in the data store **115**, for example, is associated with the operation of the various applications and/or functional entities described below.

The components executed on the computing device **103**, for example, include an application distribution service **118**, a code profiling service **121**, an electronic commerce system **124**, and other applications, services, processes, systems, engines, or functionality not discussed in detail herein. The application distribution service **118** is configured to generate and distribute limited versions of applications to clients **106**. In addition to limited versions, the application distribution service **118** may offer full versions of the applications for download. For example, the application distribution service **118** may offer applications for sale through an application marketplace, including multiple applications from multiple developers. The limited versions of the applications may be configured for execution within a secured environment of the client **106** that prevents unauthorized use of the limited version of the application and also prevents access by the limited version of the application to secured resources of the client **106**.

A code profiling service **121** may be employed by the application distribution service **118** to determine portions of the application that may be excluded from the limited version. Additionally, the application distribution service **118** may employ code profiles generated by the code profiling service **121** to reorder the code in the limited version of the application. This reordering is performed to enable intelligent downloading, wherein data is sent according to a predicted sequence of use or execution. Further, the data sent to the client **106** may be configured for a progressive download so that the limited version of the application may be executed while it is still being downloaded.

The electronic commerce system **124** is executed in order to facilitate the online purchase of items such as applications over the network **112**. The electronic commerce system **124** also performs various backend functions associated with the online presence of a merchant in order to facilitate the online purchase of items. For example, the electronic commerce system **124** may generate network pages or portions thereof that are provided to clients **106** for the purposes of selecting items for purchase, rental, download, lease, or other forms of consumption.

The data stored in the data store **115** includes, for example, applications **127**, limited applications **130**, developer configuration **133**, user configuration **136**, client profiles **139**, code profiles **142**, secured environment code **145**, simulated resource data **148**, digital rights management code **151**, application marketplace data **154**, and potentially other data. The applications **127** correspond to full versions of applications that have been uploaded by developers to be offered in an application marketplace. The applications **127** may correspond, for example, to games or other types of applications. As non-limiting examples, the application **127** may correspond to a high twitch-action game, a first-person shooter game, an action game, an adventure game, a party game, a role-playing game, a simulation game, a strategy game, a vehicle simulation game, and/or other types of games.

The applications **127** may also correspond to mobile phone applications, computer-aided design (CAD) applications, computer-aided manufacturing (CAM) applications, photo manipulation applications, video editing applications, office productivity applications, operating systems and associated applications, emulators for operating systems, architectures, and capabilities not present on a consumer device, and other applications and combinations of applications. The application **127** may be configured for execution in a general-purpose computing device or in a specialized device such as, for example, a smartphone, a video game console, a handheld game device, an arcade game device, etc. In various embodiments, the applications **127** may include support for multiple operating systems, code libraries, client computing devices, etc. which may be unnecessary depending on the client **106** on which the applications **127** are executed.

The limited applications **130** correspond to limited versions of the applications **127** which have been automatically generated by the application distribution service **118**. The limited applications **130** may have a smaller data size than the applications **127** and may exclude data and functionality which is unnecessary for evaluation use. For example, data in the corresponding application **127** which is not used during the execution of the limited application **130** may be excluded from the limited application **130**. Also, portions of the application **127** may be excluded from the limited application **130** if it is unlikely or impossible for them to be accessed or executed during a time-limited trial.

The developer configuration **133** includes parameters specified by a developer regarding its application **127**. For example, the developer configuration **133** may identify portions of the application **127** which may be excluded from the limited application **130**, features which are disabled in the limited application **130**, which portions of the application **127** are to be downloaded first to "bootstrap" execution, resources of the clients **106** that are used by the application **127**, resources of the clients **106** that should be simulated for use by the limited application **130**, resources of the client **106** which are to be protected from use by the limited applications **130**, resource of the client **106** to which the limited applications **130** are allowed access, time limits for evaluation use of the limited application **130**, restrictions on distribution of the limited application **130** and the application **127**, license terms, prices for the application **127** in the application marketplace, and so on.

The user configuration **136** includes parameters specified by, or inferred about, users or customers of the application marketplace. For example, the user configuration **136** may describe an operating system and/or other configuration characteristics of the client **106** of the user, resources of the client **106** that should be simulated for use by the limited application **130**, resources of the client **106** which are to be protected from use by the limited applications **130**, resources of the client **106** to which the limited applications **130** are allowed access, and so on. The user configuration **136** may also include account information, payment instruments, addresses, contact information, and other information used by the electronic commerce system **124**.

The client profiles **139** include data describing requirements of particular configurations of clients **106**. As a non-limiting example, the client profiles **139** may include data noting dependencies of libraries, etc. for specific operating systems or configurations. The client profiles **139** may also indicate data, code, etc. which is not used for specific operating systems or configurations. Such information may be leveraged by the application distribution service **118** to exclude certain portions of the applications **127** from the limited application **130** when the portions are not actually needed for the specific configuration of a client **106**.

The code profiles **142** may correspond to data generated by the code profiling service **121** for a limited application **130** or an application **127**. The code profiles **142** may identify sequences of code or data access for the limited application **130** or application **127**. The code profiles **142** may also indicate portions of the limited application **130** or application **127** which may possibly be reached within a given time period.

The secured environment code **145** corresponds to code that is configured to implement a secured environment **157** in a client **106**. The secured environment **157** is configured to secure one or more resources of the client **106** from access by the limited application **130**. The secured environment **157** may also be configured to prevent unauthorized use or copying of the limited application **130** and/or application **127** in the client **106**. The secured environment code **145** may include, for example, a virtual machine or other execution wrapper. The simulated resource data **148** comprises data that may be employed by the secured environment code **145** to simulate a secured resource of the client **106** for a limited application **130**. As a non-limiting example, where a list of contacts is a secured resource, the simulated resource data **148** may include a dummy list of contacts.

The digital rights management code **151** may be used in conjunction with the secured environment code **145** to prevent unauthorized use or copying of the limited application **130** and/or application **127** in the client **106**. The application marketplace data **154** includes varied forms of data used in connection with an application marketplace that offers multiple applications **127** from multiple developers. To this end, the application marketplace data **154** may include network page data, code, images, text, audio, video, etc. relating to merchandising the applications **127** and offering evaluations of the applications **127** through the use of the limited applications **130**.

The client **106** is representative of a plurality of client devices that may be coupled to the network **112**. The client **106** may comprise, for example, a processor-based system such as a computer system. Such a computer system may be embodied in the form of a desktop computer, a laptop computer, personal digital assistants, cellular telephones, smart-

phones, set-top boxes, music players, web pads, tablet computer systems, game consoles, electronic book readers, or other devices with like capability. The client **106** may include a display **160**. The display **160** may comprise, for example, one or more devices such as cathode ray tubes (CRTs), liquid crystal display (LCD) screens, gas plasma-based flat panel displays, LCD projectors, touchscreens, or other types of display devices, etc. The client **106** may include other input devices **161** and output devices **162**. The input devices **161** may comprise, for example, devices such as keyboards, mice, joysticks, accelerometers, light guns, game controllers, touch pads, touch sticks, push buttons, optical sensors, microphones, webcams, and/or any other devices that can provide user input. The output devices **162** may correspond to light indicators, haptic devices, force feedback devices, vibration devices, sound devices, and/or other devices.

The client **106** may be configured to execute various applications such as a client application **163**, a secured environment **157**, and/or other applications. The client **106** may be configured to execute applications beyond the client application **163** and the secured environment **157** such as, for example, mobile applications, email applications, instant message applications, and/or other applications. The client application **163** may correspond, for example, to a browser or mobile application that accesses and renders network pages, such as web pages, or other network content served up by the computing device **103** and/or other servers, thereby generating a screen **166** on the display **160**.

The secured environment **157** corresponds to secured environment code **145** that is configured to download and execute limited applications **130**. The secured environment **157** may incorporate a digital rights management (DRM) layer **169** that enforces trial period length restrictions, copying restrictions, and/or other types of DRM restrictions. The secured environment **157** may be further configured to download the application **127** corresponding to the limited application **130** while the limited application **130** is in use or at another time. In one embodiment, the secured environment **157** may be configured merely to download data used to transform the limited application **130** into the full application **127**. The secured environment **157** may be configured to protect various secured resources **170** of the client **106** from access by the limited application **130**. Such secured resources **170** may include, for example, contacts, email access, text message access, access to input devices **161**, access to output devices **162**, and so on.

The developer client **109** is representative of a plurality of client devices that may be coupled to the network **112**. The developer client **109** may comprise, for example, a processor-based system such as a computer system. Such a computer system may be embodied in the form of a desktop computer, a laptop computer, personal digital assistants, cellular telephones, smartphones, set-top boxes, music players, web pads, tablet computer systems, game consoles, electronic book readers, or other devices with like capability. The developer client **109** may include a display comprising, for example, one or more devices such as cathode ray tubes (CRTs), liquid crystal display (LCD) screens, gas plasma-based flat panel displays, LCD projectors, touchscreens, or other types of display devices, etc.

The developer client **109** may be configured to execute various applications such as a client application **172** and/or other applications. The developer client **109** may be configured to execute applications beyond the client application **172** such as, for example, mobile applications, email applications, instant message applications, and/or other applications. The client application **172** may correspond, for example, to a browser or mobile application that accesses and renders network pages, such as web pages, or other network content served up by the computing device **103** and/or other servers. The client application **172** may be employed, for example, by a developer to upload applications **127** and developer configurations **133** to the computing device **103**.

Next, a general description of the operation of the various components of the networked environment **100** is provided. To begin, a developer at a developer client **109** uploads an application **127** to the application distribution service **118** over the network **112**. For example, the developer may access a user interface such as a network page using the client application **172**. The developer may enter various configuration settings specifying how much the application **127** should cost in the application marketplace, distribution restrictions, evaluation version restrictions, secured resources **170** of clients **106** to which an evaluation version should not have access, and so on. Such configuration settings may be stored in the developer configuration **133**.

The application **127** is added to the application marketplace. The code profiling service **121** may process the application **127** to determine various code profiles **142**, e.g., determining which portions of the application **127** are capable of being accessed during time-limited trials, determining a sequence of code access for intelligent downloading, and so on. Users at clients **106** may register for access to the application marketplace and may specify various settings through a user interface of the client application **163**, e.g., secured resources **170** of clients **106** to which an evaluation version should not have access, information about clients **106** of the user, etc. Such settings may be stored in the user configuration **136**.

The application distribution service **118** automatically generates the limited applications **130** from the applications **127**. The limited applications **130**, which may be referred to as demonstration versions, are reduced in size compared to the applications **127** and may be generated based at least in part on an expected use of the limited application **130** during a testing period. The limited applications **130** may be customized based on the client **106** to exclude data which is unnecessary for execution in the particular client **106**. For example, the limited application **130** may exclude at least one portion of the application **127** which pertains to a different configuration from the configuration of the client **106**. The limited application **130** may be generated in response to a user request for the limited application **130** or at some other time.

In some embodiments, the application distribution service **118** may employ an intelligent downloading approach. Under such an approach, the data comprising the limited application **130** may be reordered according to a predicted code execution sequence. Such a predicted code execution sequence may be determined through code profiling of the application **127** or limited application **130** by the code profiling service **121**. Upon reordering, the limited application **130** may be sent to the client **106** so that portions that are to be executed first are sent first. Additionally, a progressive downloading approach may be employed so that the limited application **130** may begin executing in the client **106** before the limited application **130** has been received in its entirety.

Secured environment code **145** is also sent to the client **106** and may be applicable for multiple limited applications **130**. The secured environment code **145** is executed in the client **106** as a secured environment **157**. In various embodiments, the secured environment **157** may incorporate a virtual machine such as a Java Virtual Machine (JVM), a Dalvik Virtual Machine, and so on. The secured environment **157**

may be configured to facilitate progressive downloading of the limited application **130** from the application distribution service **118**.

Also, the secured environment **157** may be configured to deny access by the limited application **130** to various secured resources **170** of the client **106** as specified, for example, in the user configuration **136**, in the developer configuration **133**, at runtime by the user, etc. In some situations, the secured environment **157** may be configured to simulate the secured resources **170** using the simulated resource data **148**. As a non-limiting example, where a list of contacts corresponds to a secured resource **170**, the secured environment **157** may be configured to provide access to a dummy list of contacts for the limited application **130**. The specification of such secured resources **170** may be sent from the computing device **103** to the client **106**.

The secured environment **157** may include a DRM layer **169** that is configured to prevent unauthorized use of the limited application **130** such as unauthorized copying, running the limited application **130** beyond a predetermined number of times, beyond a cumulative time limit, or beyond a maximum instance running time, etc. In some embodiments, the secured environment **157** may be configured to obtain data relating to the application **127** to make the application **127** instantly available upon purchase. To this end, the secured environment **157** may obtain the application **127** data in the background after the limited application **130** is obtained. The DRM layer **169** may protect the application **127** data from unauthorized use before purchase.

Although it is noted that the limited applications **130** may correspond to demonstration versions of applications **127**, the limited applications **130** may also correspond to other types of limited or processed versions of applications **127**. For example, a limited application **130** may correspond to a sanitized version of an application **127** that excludes harmful code such as computer viruses, spyware, adware, and/or other malware. Such a sanitized version of the application **127** may also exclude code that has defects (e.g., code that causes memory leaks, infinite loops, etc.). Such malware and defects may be specified by a developer, application marketplace administrator, or another user. Alternatively, the malware and/or defects may be automatically identified by way of code profiling by the code profiling service **121**.

In some cases, a sanitized version of an application **127** may be generated based at least in part on a client profile **139**. As a non-limiting example, a client **106** may execute a particular version of an operating system with which an application **127** has an incompatibility. The application distribution service **118** may be configured to recognize this incompatibility and to generate a corresponding limited application **130** for the client **106** that excludes or otherwise corrects the code that causes the incompatibility. The application distribution service **118** may also configure the secured environment **157** automatically to prevent access to secured resources **170** relating to an identified problem. In some embodiments, an application **127** may be sanitized to exclude obscene language or other undesirable portions.

Turning now to FIG. **2**, shown is one example of a client **106** in the networked environment **100** (FIG. **1**). In this non-limiting example, the client **106** corresponds to a smartphone. The screen **166** of the client **106** shows a first user interface **203** generated by a limited application **130** (FIG. **1**) and a second user interface **206** generated by a secured environment **157** (FIG. **1**). The first user interface **203** corresponds to a game interface for a limited application **130** that is a racing game. Although the application **127** (FIG. **1**) from which the limited application **130** is derived may ordinarily consume the entire screen **166** for the first user interface **203**, here, the first user interface **203** shares screen **166** space with the second user interface **206**.

In this non-limiting example, the second user interface **206** is a menu bar at the top of the screen **166**. The second user interface **206** may correspond to an overlay, a side bar, a box, an interface element that is hidden at times, etc. in other examples. The second user interface **206** may include, for example, a title indicator **209**, a status indicator **212**, a purchase component **215**, an exit component **218**, and/or other components. The title indicator **209** may indicate to the user that an evaluation or trial version is being executed, that a secured environment **157** is being used, a name of an application marketplace, and/or other information. The status indicator **212** may indicate various statuses associated with the execution of the limited application **130** such as time remaining, remaining number of times the limited application **130** can be run, whether features are restricted, and so on. Here, the limited application **130** is a time-limited trial, and the status indicator **212** shows that nine minutes remain in the trial.

The purchase component **215** may be linked with the electronic commerce system **124** (FIG. **1**) so that, when selected, the purchase component **215** initiates a purchase in the application marketplace of the full application **127** that corresponds to the limited application **130**. Alternatively, the purchase component **215** may be configured to bring up a user interface in the client application **163**, such as a rendered network page, with more information about ordering the full application **127** through the application marketplace. The exit component **218** may be used to terminate the limited application **130** and exit the secured environment **157**. Different components, fewer components, or other components may be present in other examples of the second user interface **206**.

Moving on to FIG. **3**, shown is a flowchart that provides one example of the operation of a portion of the application distribution service **118** according to various embodiments. It is understood that the flowchart of FIG. **3** provides merely an example of the many different types of functional arrangements that may be employed to implement the operation of the portion of the application distribution service **118** as described herein. As an alternative, the flowchart of FIG. **3** may be viewed as depicting an example of steps of a method implemented in the computing device **103** (FIG. **1**) according to one or more embodiments.

Beginning with box **303**, the application distribution service **118** obtains the application **127** (FIG. **1**) by way of the network **112** (FIG. **1**) from the developer of the application **127** at the developer client **109** (FIG. **1**). The developer may also provide various configuration settings for the developer configuration **133** (FIG. **1**). In box **306**, the application distribution service **118** adds the application **127** to the application marketplace. In box **309**, the application distribution service **118** obtains a request from a client **106** (FIG. **1**) to download a limited version of the application **127**. For example, the application **127** may be offered as a free, time-limited or feature-limited trial through the application marketplace.

In box **312**, the application distribution service **118** determines whether the client **106** has the appropriate secured environment **157** (FIG. **1**). If the client **106** does not have the appropriate secured environment **157**, then in box **315**, the application distribution service **118** sends secured environment code **145** (FIG. **1**) to the client **106** over the network **112**. In various embodiments, at least a portion of the secured environment code **145** is applicable to execute limited versions of different applications **127** offered in the application

marketplace. The secured environment code **145** may be customized according to characteristics of the client **106** such as, for example, device type, operating system, existing software libraries, and so on. The application distribution service **118** continues to box **318**. If the application distribution service **118** determines that the client **106** already has a secured environment **157**, the application distribution service **118** proceeds from box **312** to box **318**.

In box **318**, the application distribution service **118** automatically generates a limited application **130** (FIG. **1**) from the application **127**. The limited application **130** may have a reduced data size compared to the application **127**. In some embodiments, the limited application **130** may have previously been generated and stored in the data store **115** (FIG. **1**). The limited application **130** may be generated based at least in part on an expected use of the application **127** by the client **106** during a testing period. For example, the limited application **130** may exclude a portion of the application **127** that is predicted to be unaccessed during the testing or evaluation period. Such a portion may remain unaccessed because a time limitation does not allow for a user to reach functionality of the portion, the client **106** does not support the portion, or for other reasons. The code profiles **142** may indicate which portions of the application **127** are predicted to remain unaccessed.

In box **321**, the application distribution service **118** sends the limited application **130** to the client **106**. The limited application **130** may be sent using an intelligent download approach and/or a progressive download approach so that the secured environment **157** of the client **106** is able to commence execution of the limited application **130** before the limited application **130** has finished downloading. In box **324**, the application distribution service **118** determines whether the client **106** has requested the full version of the application **127**. For example, the user at the client **106** may select a purchase component **215** (FIG. **2**) or other user interface component that corresponds to a request for the full version of the application **127**. In some embodiments, the secured environment **157** may be configured to request the full version of the application **127** automatically so long as the download does not diminish the user experience due to bandwidth on the network **112**, consumed resources of the client **106**, and so on.

If the full version of the application **127** is to be sent, the application distribution service **118** proceeds to box **327** and sends the full version of the application **127** to the client **106**. In one embodiment, the sent data corresponds to data for transforming the limited application **130** into the full application **127** or some other subset of data that is less than the entire application **127**. The data that is sent may be customized according to the capabilities of the client **106**. Thereafter, the portion of the application distribution service **118** ends. If the full version of the application **127** is not to be sent in box **324**, the portion of the application distribution service **118** also ends.

Referring next to FIG. **4**, shown is a flowchart that provides one example of the operation of a portion of the secured environment **157** according to various embodiments. It is understood that the flowchart of FIG. **4** provides merely an example of the many different types of functional arrangements that may be employed to implement the operation of the portion of the secured environment **157** as described herein. As an alternative, the flowchart of FIG. **4** may be viewed as depicting an example of steps of a method implemented in the client **106** (FIG. **1**) according to one or more embodiments.

Beginning with box **403**, the secured environment **157** determines the secured resources **170** (FIG. **1**) of the client **106**. For example, the secured environment **157** may provide a user interface that prompts a user to select various resources as secured or non-secured. Such resources may correspond to contact lists, telephone capability, global positioning system (GPS) information, text message capability, email capability, and so on. The secured environment **157** may design various resources as secured by default.

In box **406**, the secured environment **157** begins a download of a limited application **130** (FIG. **1**) from the application distribution service **118** (FIG. **1**). For example, the secured environment **157** may be preconfigured to download the limited application **130** as a result of the user selecting a link in a user interface of the client application **163** (FIG. **1**) or otherwise indicating a selection of a limited application **130**. The download may be performed, for example, using an intelligent download and/or progressive download approach.

In box **409**, the secured environment **157** determines whether the limited application **130** can begin executing. For example, sufficient data may have been received to begin execution of the limited application **130** according to a progressive download approach. If sufficient data has not been received, the secured environment **157** moves to box **412** and continues downloading the limited application **130** from the application distribution service **118**. The secured environment **157** then returns to box **409**. If the secured environment **157** instead determines in box **409** that sufficient data has been received, the secured environment **157** proceeds to box **415**.

In box **415**, the secured environment **157** begins execution of the limited application **130**, potentially while the download of the limited application **130** continues under a progressive download approach. While the limited application **130** executes, the secured environment **157** may deny access by the limited application **130** to secured resources **170**. Alternatively, the secured environment **157** may provide access to simulated versions of the secured resources **170** in the client **106**. In box **418**, the secured environment **157** completes the download of the limited application **130**. In box **421**, the secured environment **157** determines if a time limit has been reached, or if the limited application **130** has been exited. If a time limit has not been reached, or if the limited application **130** has not been exited, the secured environment **157** returns to box **421** while the execution of the limited application **130** continues. If a time limit has been reached, or if the limited application **130** has been exited, the secured environment **157** proceeds to box **424**.

In box **424**, the secured environment **157** terminates execution of the limited application **130**. In box **427**, the secured environment **157** determines whether the full version of the application **127** (FIG. **1**) corresponding to the limited application **130** is to be obtained. If the full version of the application **127** is to be obtained, the secured environment **157** transitions to box **430** and downloads the full version of the application **127** from the application distribution service **118**. In some cases, the data obtained for the full version may be merely the data to transform the limited application **130** into the application **127**. Such data may be maintained in a secured state by the DRM layer **169** (FIG. **1**) until unlocked or purchased. Thereafter, the portion of the secured environment **157** ends. If the full version is not to be obtained in box **327**, the portion of the secured environment **157** also ends.

With reference to FIG. **5**, shown is a schematic block diagram of the computing device **103** according to an embodiment of the present disclosure. The computing device **103** includes at least one processor circuit, for example, hav-

ing a processor **503** and a memory **506**, both of which are coupled to a local interface **509**. To this end, the computing device **103** may comprise, for example, at least one server computer or like device. The local interface **509** may comprise, for example, a data bus with an accompanying address/control bus or other bus structure as can be appreciated.

Stored in the memory **506** are both data and several components that are executable by the processor **503**. In particular, stored in the memory **506** and executable by the processor **503** are the application distribution service **118**, the code profiling service **121**, the electronic commerce system **124**, and potentially other applications. Also stored in the memory **506** may be a data store **115** and other data. In addition, an operating system may be stored in the memory **506** and executable by the processor **503**.

It is understood that there may be other applications that are stored in the memory **506** and are executable by the processor **503** as can be appreciated. Where any component discussed herein is implemented in the form of software, any one of a number of programming languages may be employed such as, for example, C, C++, C#, Objective C, Java®, JavaScript®, Perl, PHP, Visual Basic®, Python®, Ruby, Delphi®, Flash®, or other programming languages.

A number of software components are stored in the memory **506** and are executable by the processor **503**. In this respect, the term "executable" means a program file that is in a form that can ultimately be run by the processor **503**. Examples of executable programs may be, for example, a compiled program that can be translated into machine code in a format that can be loaded into a random access portion of the memory **506** and run by the processor **503**, source code that may be expressed in proper format such as object code that is capable of being loaded into a random access portion of the memory **506** and executed by the processor **503**, or source code that may be interpreted by another executable program to generate instructions in a random access portion of the memory **506** to be executed by the processor **503**, etc. An executable program may be stored in any portion or component of the memory **506** including, for example, random access memory (RAM), read-only memory (ROM), hard drive, solid-state drive, USB flash drive, memory card, optical disc such as compact disc (CD) or digital versatile disc (DVD), floppy disk, magnetic tape, or other memory components.

The memory **506** is defined herein as including both volatile and nonvolatile memory and data storage components. Volatile components are those that do not retain data values upon loss of power. Nonvolatile components are those that retain data upon a loss of power. Thus, the memory **506** may comprise, for example, random access memory (RAM), read-only memory (ROM), hard disk drives, solid-state drives, USB flash drives, memory cards accessed via a memory card reader, floppy disks accessed via an associated floppy disk drive, optical discs accessed via an optical disc drive, magnetic tapes accessed via an appropriate tape drive, and/or other memory components, or a combination of any two or more of these memory components. In addition, the RAM may comprise, for example, static random access memory (SRAM), dynamic random access memory (DRAM), or magnetic random access memory (MRAM) and other such devices. The ROM may comprise, for example, a programmable read-only memory (PROM), an erasable programmable read-only memory (EPROM), an electrically erasable programmable read-only memory (EEPROM), or other like memory device.

Also, the processor **503** may represent multiple processors **503** and the memory **506** may represent multiple memories **506** that operate in parallel processing circuits, respectively. In such a case, the local interface **509** may be an appropriate network that facilitates communication between any two of the multiple processors **503**, between any processor **503** and any of the memories **506**, or between any two of the memories **506**, etc. The local interface **509** may comprise additional systems designed to coordinate this communication, including, for example, performing load balancing. The processor **503** may be of electrical or of some other available construction.

Although the application distribution service **118**, the code profiling service **121**, the electronic commerce system **124**, the secured environment **157** (FIG. **1**), the client application **163** (FIG. **1**), the client application **172** (FIG. **1**), and other various systems described herein may be embodied in software or code executed by general purpose hardware as discussed above, as an alternative the same may also be embodied in dedicated hardware or a combination of software/general purpose hardware and dedicated hardware. If embodied in dedicated hardware, each can be implemented as a circuit or state machine that employs any one of or a combination of a number of technologies. These technologies may include, but are not limited to, discrete logic circuits having logic gates for implementing various logic functions upon an application of one or more data signals, application specific integrated circuits having appropriate logic gates, or other components, etc. Such technologies are generally well known by those skilled in the art and, consequently, are not described in detail herein.

The flowcharts of FIGS. **3** and **4** show the functionality and operation of an implementation of portions of the application distribution service **118** and the secured environment **157**. If embodied in software, each block may represent a module, segment, or portion of code that comprises program instructions to implement the specified logical function(s). The program instructions may be embodied in the form of source code that comprises human-readable statements written in a programming language or machine code that comprises numerical instructions recognizable by a suitable execution system such as a processor **503** in a computer system or other system. The machine code may be converted from the source code, etc. If embodied in hardware, each block may represent a circuit or a number of interconnected circuits to implement the specified logical function(s).

Although the flowcharts of FIGS. **3** and **4** show a specific order of execution, it is understood that the order of execution may differ from that which is depicted. For example, the order of execution of two or more blocks may be scrambled relative to the order shown. Also, two or more blocks shown in succession in FIGS. **3** and **4** may be executed concurrently or with partial concurrence. Further, in some embodiments, one or more of the blocks shown in FIGS. **3** and **4** may be skipped or omitted. In addition, any number of counters, state variables, warning semaphores, or messages might be added to the logical flow described herein, for purposes of enhanced utility, accounting, performance measurement, or providing troubleshooting aids, etc. It is understood that all such variations are within the scope of the present disclosure.

Also, any logic or application described herein, including the application distribution service **118**, the code profiling service **121**, the electronic commerce system **124**, the secured environment **157**, the client application **163**, the client application **172**, that comprises software or code can be embodied in any non-transitory computer-readable medium for use by or in connection with an instruction execution system such as, for example, a processor **503** in a computer system or other system. In this sense, the logic may comprise, for example,

statements including instructions and declarations that can be fetched from the computer-readable medium and executed by the instruction execution system. In the context of the present disclosure, a "computer-readable medium" can be any medium that can contain, store, or maintain the logic or application described herein for use by or in connection with the instruction execution system.

The computer-readable medium can comprise any one of many physical media such as, for example, magnetic, optical, or semiconductor media. More specific examples of a suitable computer-readable medium would include, but are not limited to, magnetic tapes, magnetic floppy diskettes, magnetic hard drives, memory cards, solid-state drives, USB flash drives, or optical discs. Also, the computer-readable medium may be a random access memory (RAM) including, for example, static random access memory (SRAM) and dynamic random access memory (DRAM), or magnetic random access memory (MRAM). In addition, the computer-readable medium may be a read-only memory (ROM), a programmable read-only memory (PROM), an erasable programmable read-only memory (EPROM), an electrically erasable programmable read-only memory (EEPROM), or other type of memory device.

It should be emphasized that the above-described embodiments of the present disclosure are merely possible examples of implementations set forth for a clear understanding of the principles of the disclosure. Many variations and modifications may be made to the above-described embodiment(s) without departing substantially from the spirit and principles of the disclosure. All such modifications and variations are intended to be included herein within the scope of this disclosure and protected by the following claims.

Therefore, the following is claimed:

1. A non-transitory computer-readable medium embodying a program executable in at least one computing device, the program comprising:

code that obtains a plurality of applications from a plurality of developers;

code that offers the applications for sale in an application marketplace;

code that offers a demonstration version of one of the applications in the application marketplace;

code that automatically generates the demonstration version of the one of the applications from a full version of the one of the applications, wherein at least one portion of the full version is excluded from the demonstration version based at least in part on a predicted use of the demonstration version, and the demonstration version has a reduced data size compared to the full version;

code that sends code that implements a secured environment to a client computing device, wherein the secured environment is configured to execute demonstration versions of the applications in the client computing device such that the demonstration versions of the applications are denied access to at least one secured resource of the client computing device;

code that sends the demonstration version of the one of the applications to the client computing device;

wherein data corresponding to the demonstration version of the one of the applications is sent according to a sequence established based at least in part on a code access profile for the demonstration version of the one of the applications; and

wherein the client computing device is configured to commence execution of the demonstration version of the one of the applications while a portion of the demonstration version of the one of the applications remains to be received by the client computing device.

2. The non-transitory computer-readable medium of claim **1**, wherein the demonstration version of the one of the applications is automatically generated based at least in part on a configuration associated with the client computing device and in response to a download request obtained from the client computing device.

3. The non-transitory computer-readable medium of claim **1**, wherein the code that implements the secured environment is configured to enforce a maximum running time for the demonstration version of the one of the applications in the client computing device.

4. The non-transitory computer-readable medium of claim **1**, wherein the client computing device is a mobile computing device.

5. A system, comprising:

at least one computing device; and

an application distribution service executable in the at least one computing device, the application distribution service comprising:

logic that automatically generates a limited version of an application from a full version of the application based at least in part on an expected use of the application by a client computing device during a testing period, the limited version having a smaller data size than the full version; and

logic that sends the limited version of the application to the client computing device, wherein the limited version of the application is configured to be executed in a secured environment of the client computing device that denies access to at least one secured resource of the client computing device by the limited version of the application.

6. The system of claim **5**, wherein the application distribution service further comprises logic that sends simulated resource data to the client computing device, wherein the secured environment is configured to simulate the at least one secured resource for the limited version of the application according to the simulated resource data.

7. The system of claim **5**, wherein the application distribution service further comprises logic that sends code that implements the secured environment to the client computing device.

8. The system of claim **5**, wherein the secured environment is configured to obtain a user specification of the at least one secured resource to which access by the limited version of the application is to be denied.

9. The system of claim **5**, wherein the application distribution service further comprises:

logic that obtains the full version of the application from a developer of the application; and

logic that adds the application to an application marketplace that offers a plurality of applications by a plurality of developers.

10. The system of claim **9**, wherein the application distribution service further comprises:

logic that obtains a specification of the at least one secured resource to which access by the limited version of the application is to be denied from the developer; and

logic that sends the specification of the at least one secured resource to the client computing device.

11. The system of claim **5**, wherein the logic that sends the limited version of the application to the client computing device further comprises logic that reorders code corresponding to the limited version of the application based at least in part on a predicted code execution sequence.

12. The system of claim **11**, wherein the application distribution service further comprises logic that determines the predicted code execution sequence through code profiling of the application.

13. The system of claim **5**, wherein the secured environment is configured to execute the limited version of the application while a portion of the limited version of the application is being downloaded.

14. The system of claim **5**, wherein the expected use corresponds to a time-limited trial of the application, and the limited version of the application excludes at least one portion of the full version of the application which is predicted to be unaccessed during the testing period.

15. The system of claim **14**, wherein the application distribution service further comprises logic that determines the at least one portion of the full version of the application which is predicted to be unaccessed during the testing period based at least in part on code profiling of the application.

16. The system of claim **5**, wherein the expected use corresponds to execution by a first configuration of the client computing device, and the limited version of the application excludes at least one portion of the full version of the application which pertains to a second configuration of the client computing device that is different from the first configuration.

17. The system of claim **5**, wherein the expected use corresponds to execution by a configuration of the client computing device, and the limited version of the application excludes at least one portion of the full version of the application which is identified as harmful to the configuration of the client computing device.

18. The system of claim **5**, wherein the secured environment includes a digital rights management (DRM) layer that is configured to prevent unauthorized use of the limited version of the application.

19. The system of claim **18**, wherein the unauthorized use corresponds to unauthorized copying of the limited version of the application.

20. A method, comprising:

automatically generating, in at least one computing device, a limited version of an application from a full version of the application, wherein the limited version has a reduced data size compared to the full version and excludes a portion of the application that is predicted to be unaccessed during an evaluation period;

reordering, in the at least one computing device, code of the limited version of the application based at least in part on a code access sequence predicted for the limited version of the application; and

sending, in the at least one computing device, the limited version of the application to a client computing device, wherein the limited version of the application is configured to commence execution in the client computing device before the limited version of the application is sent.

21. The method of claim **20**, further comprising sending, in the at least one computing device, data for transforming the limited version of the application into the full version of the application after the limited version of the application is sent.

22. The method of claim **20**, further comprising sending, in the at least one computing device, code that implements a secured environment to the client computing device, wherein the limited version of the application is configured to be executed in the secured environment, and the secured environment is configured to deny access by the limited version of the application to at least one secured resource of the client computing device.

23. The method of claim **22**, wherein the secured environment is configured to enforce a time limit for the evaluation period.

24. The method of claim **22**, wherein the secured environment is configured to render a first user interface that includes a second user interface rendered by the limited version of the application and a component for initiating a purchase of the full version of the application.

* * * * *

Exhibit 2

US011061812B2

(12) **United States Patent**
Qureshi et al.

(10) **Patent No.: US 11,061,812 B2**
(45) **Date of Patent: Jul. 13, 2021**

(54) **USING CONTAINERS FOR UPDATE DEPLOYMENT**

(71) Applicant: **Amazon Technologies, Inc.**, Seattle, WA (US)

(72) Inventors: **Tipu Saleem Qureshi**, Seattle, WA (US); **Deepak Singh**, Issaquah, WA (US)

(73) Assignee: **Amazon Technologies, Inc.**, Seattle, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 50 days.

(21) Appl. No.: **15/914,928**

(22) Filed: **Mar. 7, 2018**

(65) **Prior Publication Data**

US 2018/0196741 A1 Jul. 12, 2018

Related U.S. Application Data

(63) Continuation of application No. 14/671,996, filed on Mar. 27, 2015, now Pat. No. 9,916,233.

(51) **Int. Cl.**
G06F 9/445 (2018.01)
G06F 11/36 (2006.01)
(Continued)

(52) **U.S. Cl.**
CPC ***G06F 11/3692*** (2013.01); ***G06F 8/60*** (2013.01); ***G06F 8/65*** (2013.01); ***G06F 8/70*** (2013.01);
(Continued)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,974,454 A * 10/1999 Apfel G06F 8/65 709/218
6,353,928 B1 * 3/2002 Altberg G06F 8/61 717/175
(Continued)

FOREIGN PATENT DOCUMENTS

WO 2014047073 A1 3/2014

OTHER PUBLICATIONS

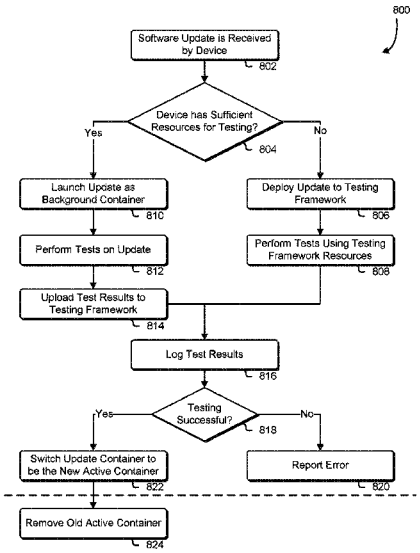
He et al., "Elastic Application Container: A Lightweight Approach for Cloud Resource Provisioning," 26th IEEE International Conference on Advanced Information Networking and Applications, Mar. 26, 2012, pp. 15-22.
(Continued)

Primary Examiner — Wei Y Zhen
Assistant Examiner — Lanny N Ung
(74) *Attorney, Agent, or Firm* — Davis Wright Tremaine LLP

(57) **ABSTRACT**

A system and method for software deployment, where the system and method include, at a deployment service, obtaining a software package and determining that a client device is ready to receive at least a portion of the software package. If the client device is ready, providing at least the portion of the software package to the client device, launching at least the provided portion of the software package as set of instructions executing in a test container, and performing a set of tests on the executing set of instructions in the test container. Based at least in part on results of performing the set of tests, determining whether to cause at least the provided portion of the software package to execute in an active container on the client device.

20 Claims, 9 Drawing Sheets



(51) **Int. Cl.**

G06F 8/65	(2018.01)
G06F 8/70	(2018.01)
G06F 9/455	(2018.01)
G06F 8/60	(2018.01)

(52) **U.S. Cl.**
CPC ***G06F 9/45533*** (2013.01); ***G06F 11/3688*** (2013.01); *G06F 9/44521* (2013.01)

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,096,464	B1 *	8/2006	Weinmann	G06F 8/65 717/168
7,539,985	B2 *	5/2009	Marvin	G06F 8/71 717/170
7,805,706	B1	9/2010	Ly et al.	
8,407,696	B2 *	3/2013	Alpern	G06F 8/65 717/168
8,621,069	B1	12/2013	Tompkins	
8,788,855	B2	7/2014	Cong et al.	
2004/0040025	A1	2/2004	Lehtinen	
2005/0060722	A1 *	3/2005	Rochette	G06F 8/60 719/319
2005/0262494	A1 *	11/2005	Fung	G06F 8/656 717/170
2006/0136928	A1	6/2006	Crawford et al.	
2007/0118657	A1	5/2007	Kreitzer et al.	
2007/0204003	A1 *	8/2007	Abramson	H04L 67/06 709/217
2010/0005455	A1 *	1/2010	Gyure	G06F 11/3636 717/128
2010/0070961	A1	3/2010	Auer et al.	
2011/0153798	A1 *	6/2011	Groenendaal	H04L 63/20 709/223
2011/0154109	A1	6/2011	Levine et al.	
2012/0124003	A1	5/2012	Sreedharan	
2013/0111461	A1 *	5/2013	Zubas	G06F 8/654 717/173
2013/0191527	A1	7/2013	Ashok et al.	
2013/0198764	A1	8/2013	Kacin et al.	
2013/0297964	A1	11/2013	Hegdal et al.	
2014/0149986	A1	5/2014	S M et al.	

OTHER PUBLICATIONS

International Search Report and Written Opinion dated Feb. 4, 2016, International Patent Application No. PCT/US2015/059983, 12 pages.

Soltesz et al., "Container-based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors," ACM SIGOPS Operating Systems Review 41(3):275-287, Mar. 2007.

Van et al., "SLA-aware Virtual Resource Management for Cloud Infrastructures," IEEE Ninth International Conference on Computer and Information Technology, Oct. 11, 2009, pp. 357-362.

Wikipedia, "Docker (Software)," Jul. 21, 2016, retrieved Jul. 29, 2016, from http://en.wikipedia.org/wiki/Docker_(software), 6 pages.

Wikipedia, "LXC," Jul. 6, 2016, retrieved Jul. 29, 2016, from http://en.wikipedia.org/wiki/LXC, 3 pages.

Xavier et al., "Performance Evaluation of Container-based Virtualization for High Performance Computing Environments," Parallel, Distributed, and Network-Based Processing (PDP), 2013 21st Euromicro International Conference, Feb. 2013, pp. 233-240.

Zhao et al., "Experimental Study of Virtual Machine Migration in Support of Reservation of Cluster Resources," Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed Computing, Nov. 2007, pp. 1-8.

* cited by examiner

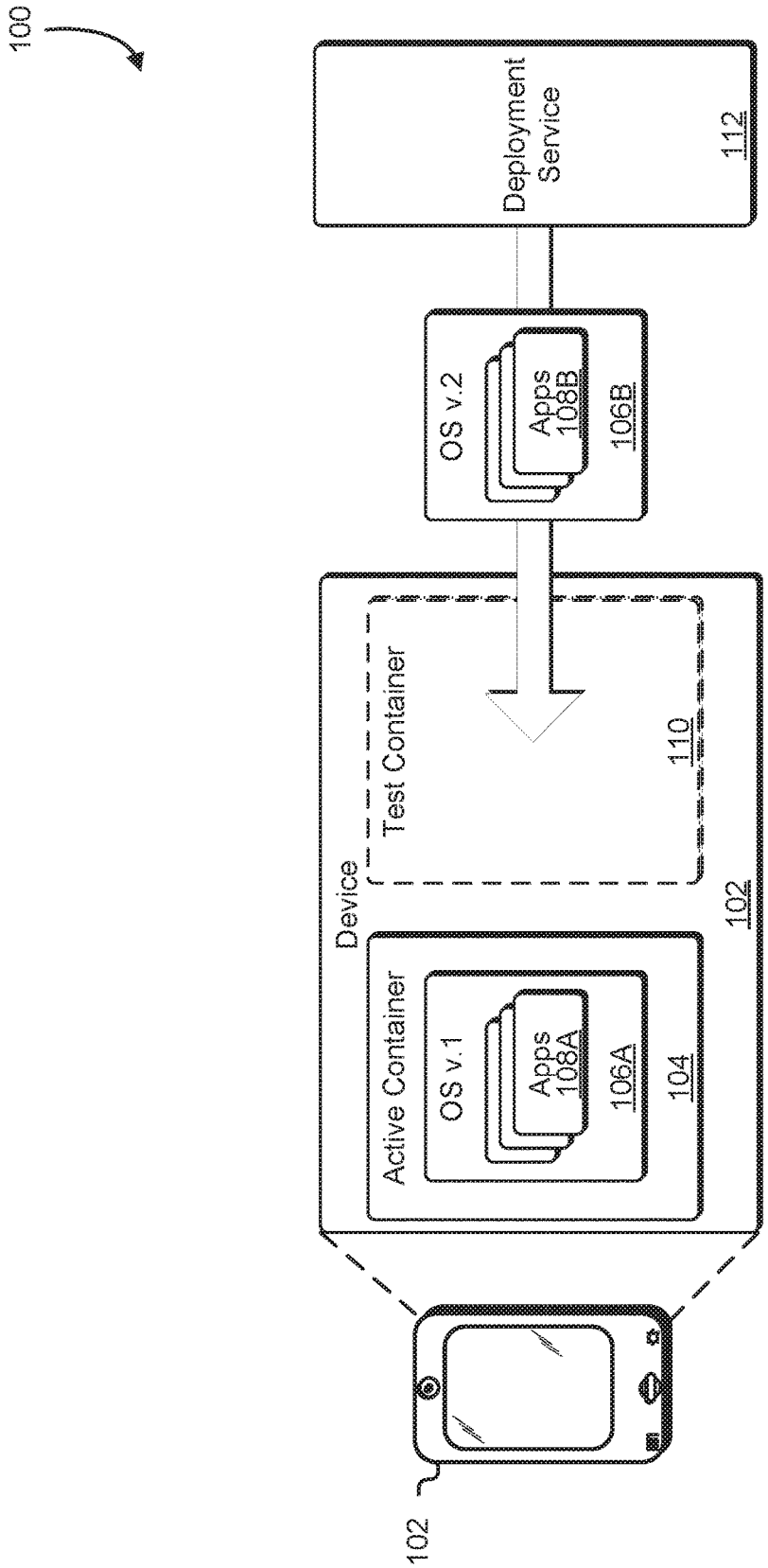

100
Deployment Service
112
OS v.2
Apps 108B
106B
Test Container
110
Device
102
Active Container
OS v.1
Apps 108A
106A
104
102

FIG. 1

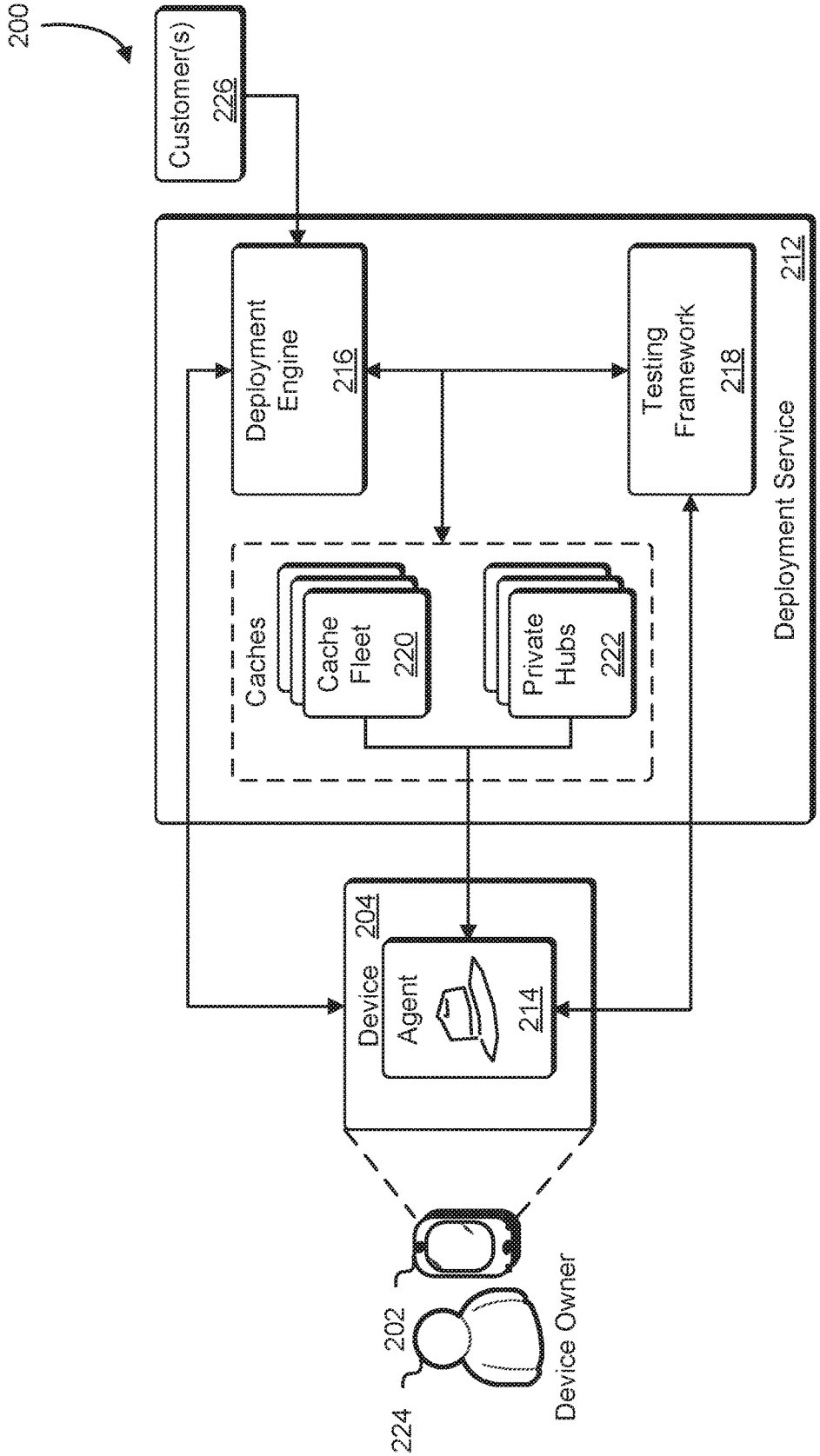

200
Customer(s)
226
Deployment Service
212
Deployment Engine
216
Testing Framework
218
Caches
Cache Fleet
220
Private Hubs
222
Device
204
Agent
214
202
Device Owner
224

FIG. 2

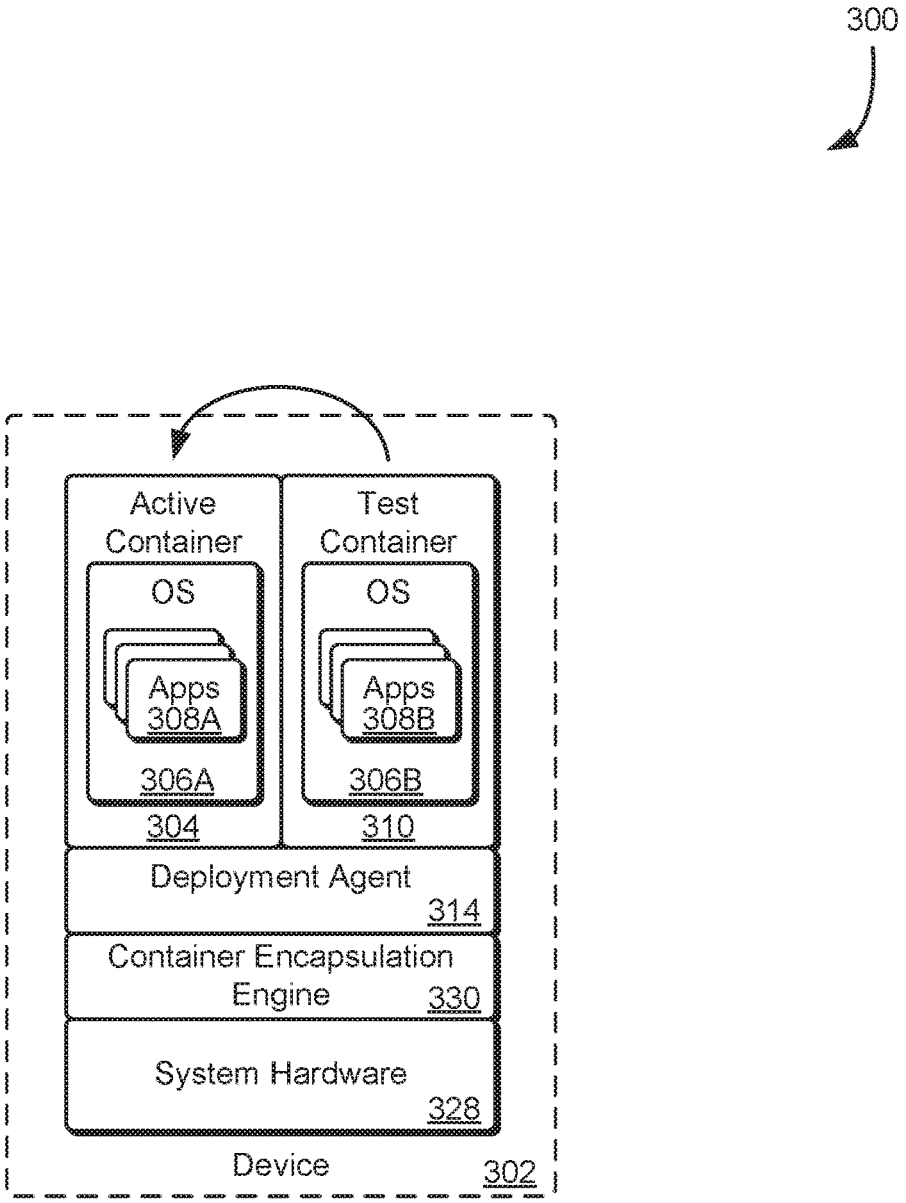

300
Active Container
OS
Apps 308A
306A
304
Test Container
OS
Apps 308B
306B
310
Deployment Agent
314
Container Encapsulation Engine
330
System Hardware
328
Device
302

FIG. 3

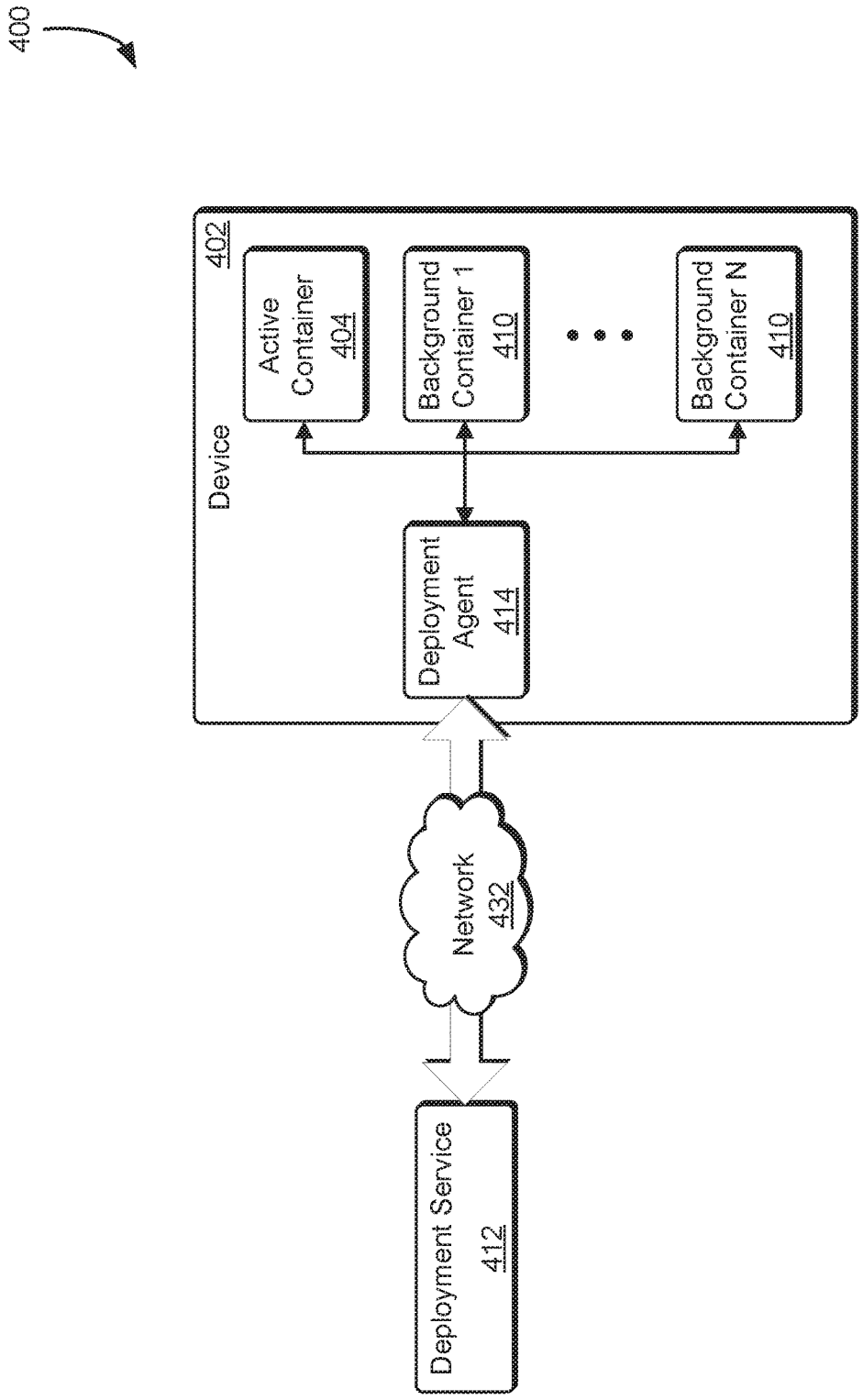

400
402
Device
Active Container 404
Background Container 1 410
Background Container N 410
Deployment Agent 414
Network 432
Deployment Service 412

FIG. 4

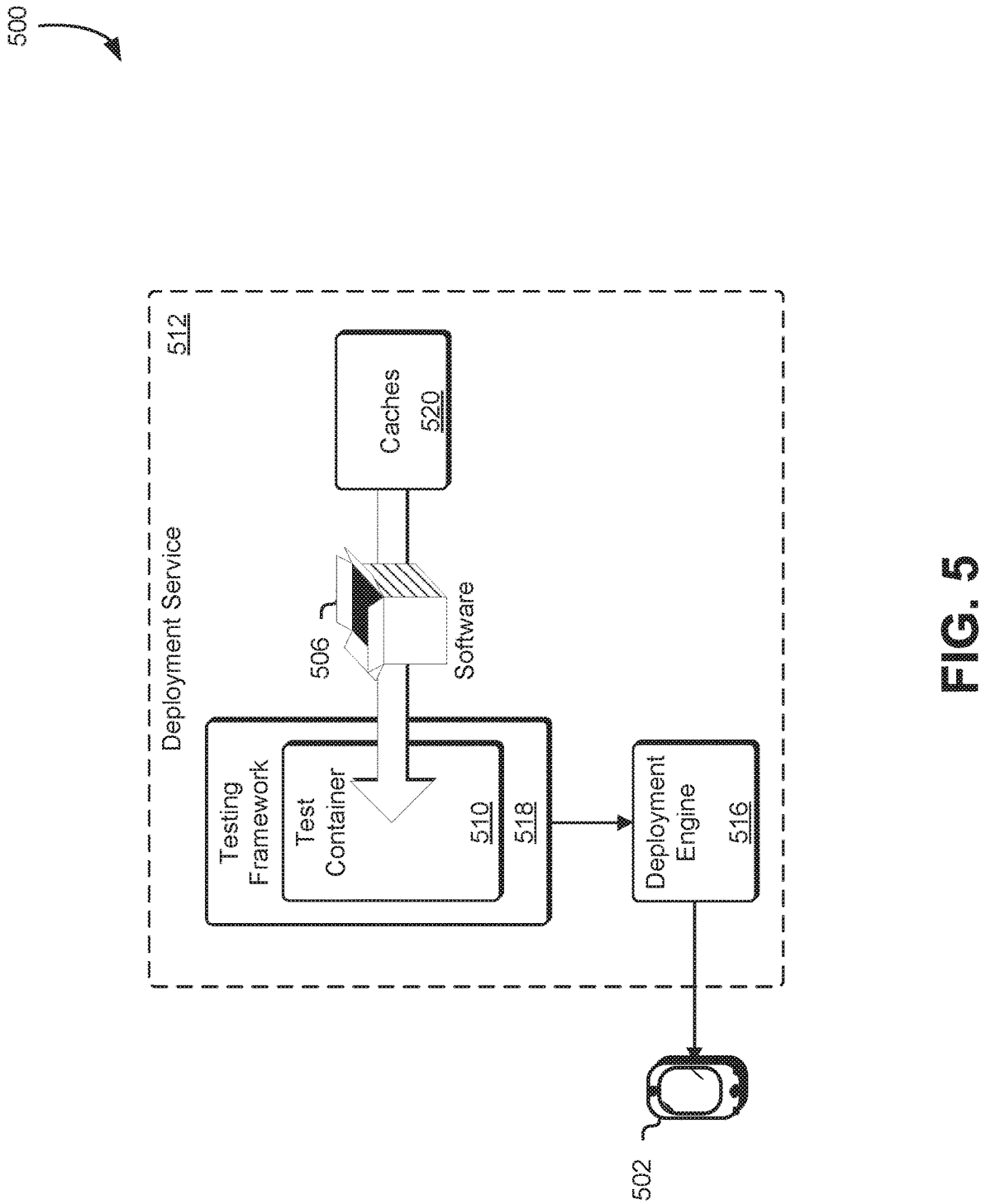

500
Deployment Service
512
Caches
520
506
Software
Testing Framework
Test Container
510
518
Deployment Engine
516
502

FIG. 5

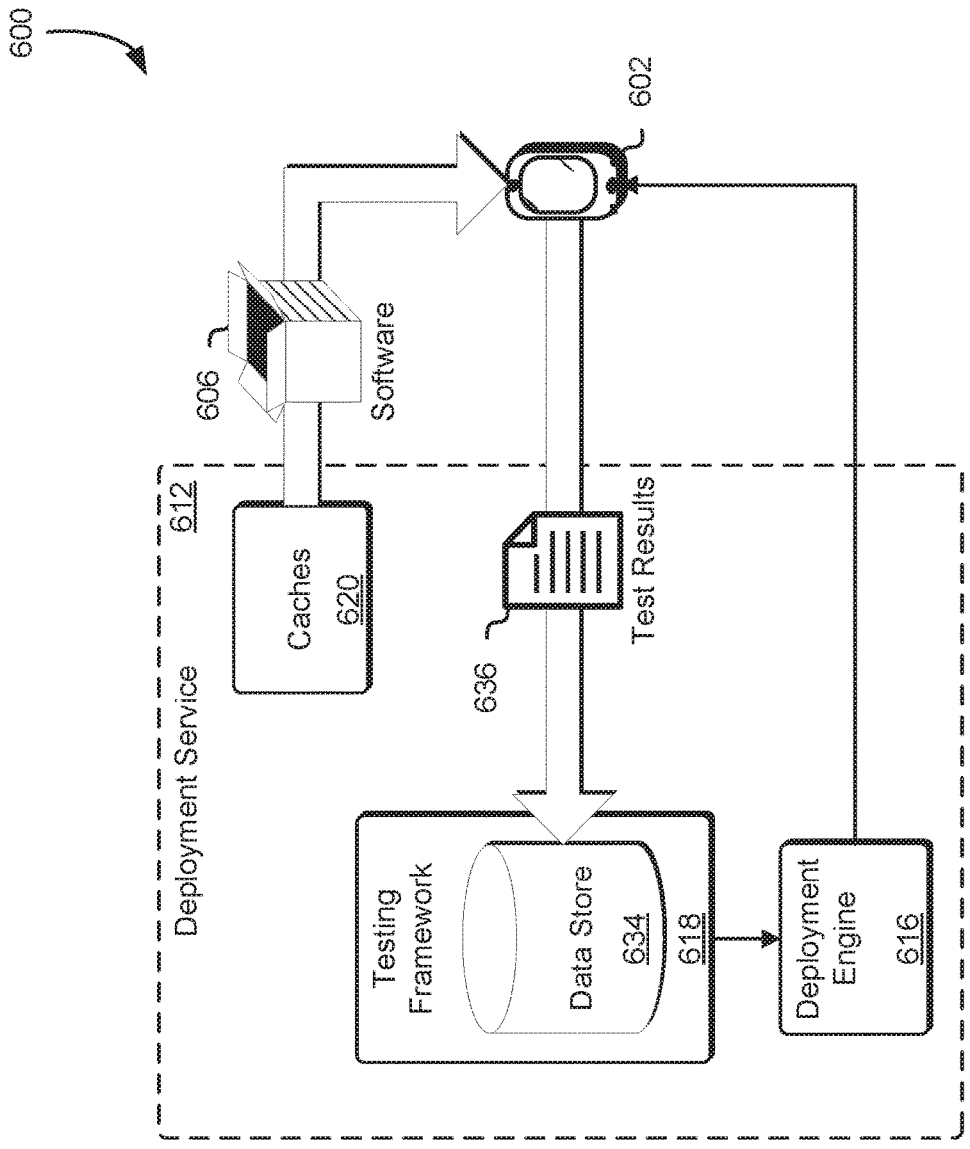

600
602
606
Software
612
Deployment Service
Caches
620
636
Test Results
Testing
Framework
Data Store
634
618
Deployment
Engine
616

FIG. 6

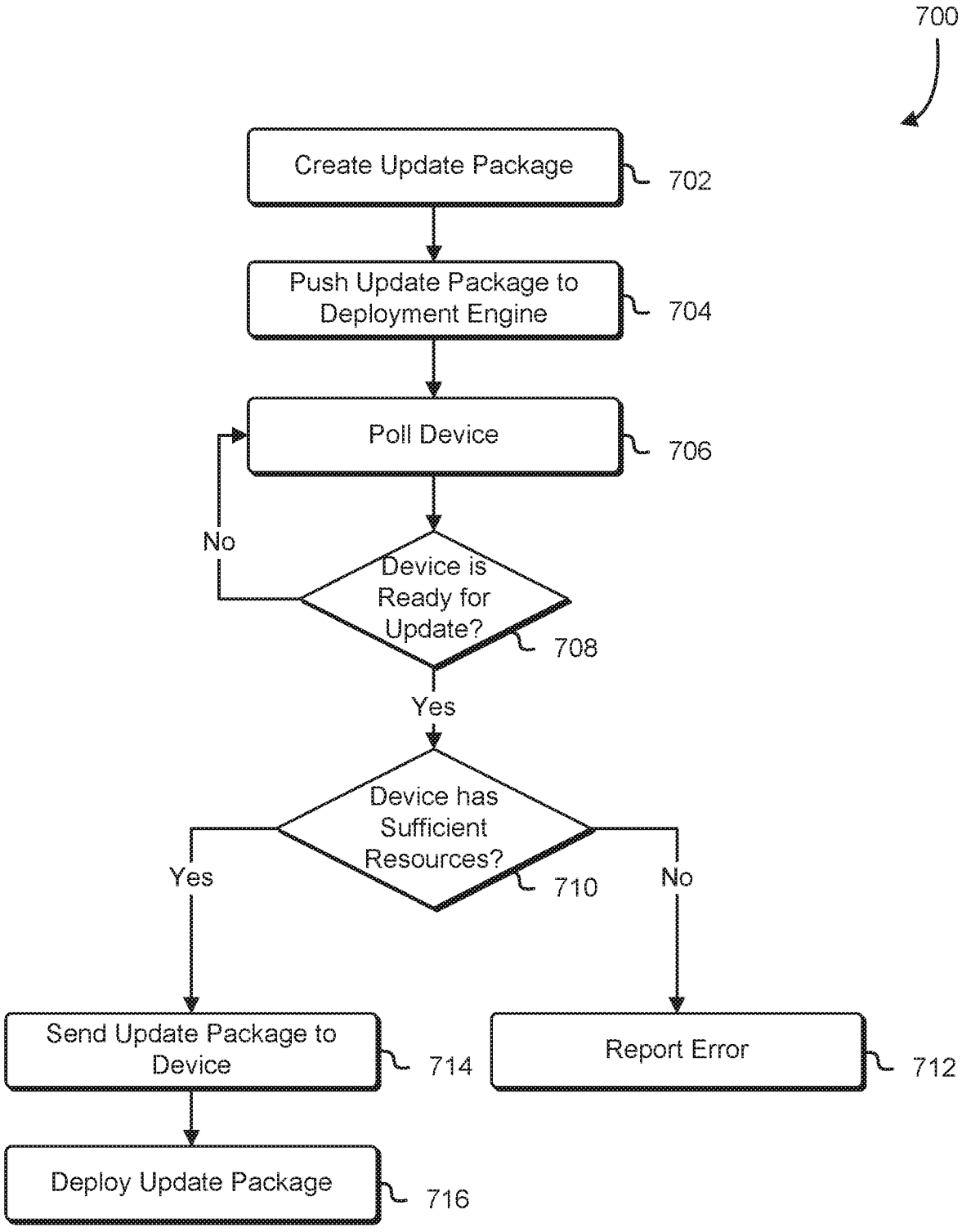

700
Create Update Package
702
Push Update Package to Deployment Engine
704
Poll Device
706
No
Device is Ready for Update?
708
Yes
Device has Sufficient Resources?
710
Yes
No
Send Update Package to Device
714
Report Error
712
Deploy Update Package
716

FIG. 7

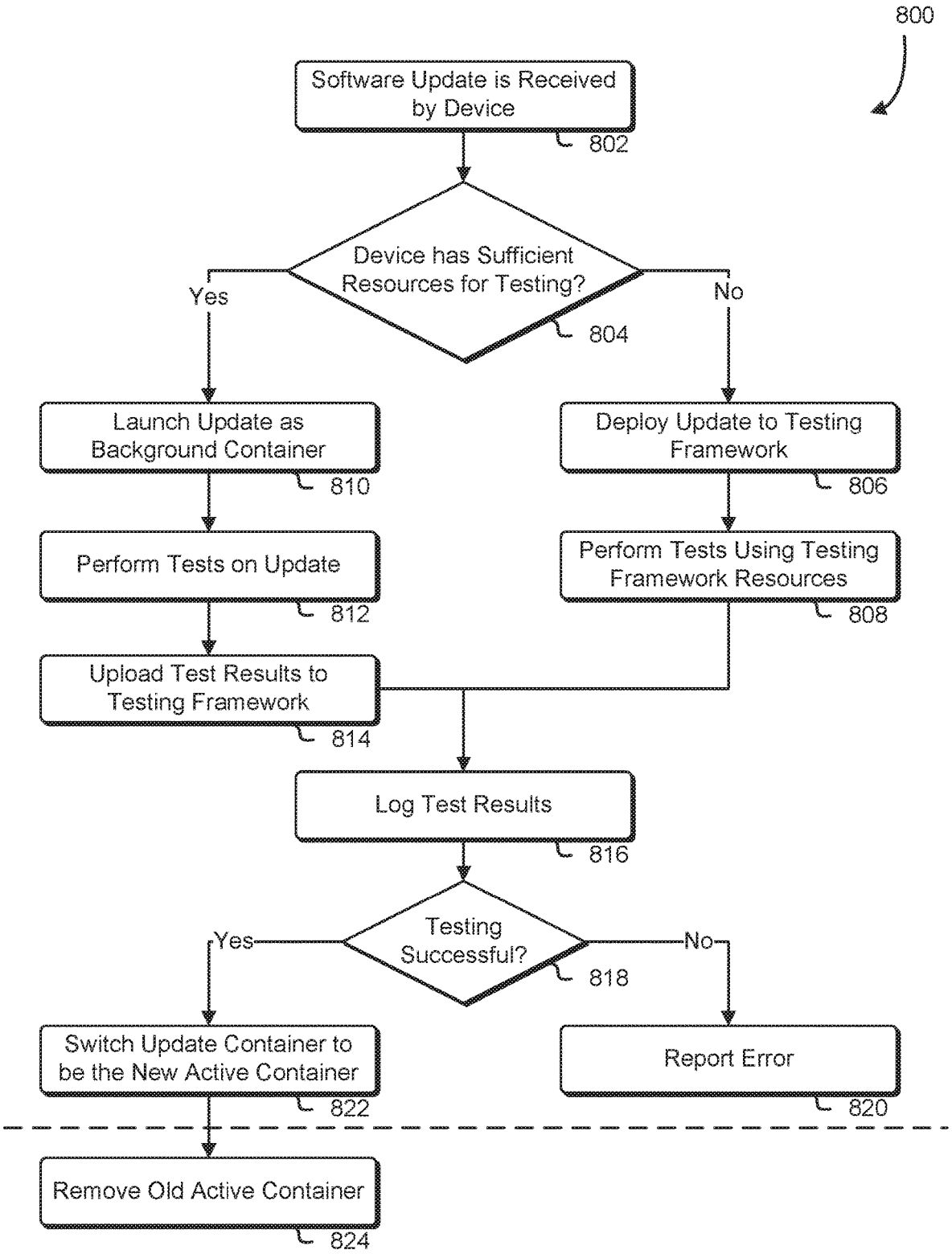

800
Software Update is Received by Device
802
Device has Sufficient Resources for Testing?
804
Yes
No
Launch Update as Background Container
810
Perform Tests on Update
812
Upload Test Results to Testing Framework
814
Deploy Update to Testing Framework
806
Perform Tests Using Testing Framework Resources
808
Log Test Results
816
Testing Successful?
818
Yes
No
Switch Update Container to be the New Active Container
822
Report Error
820
Remove Old Active Container
824

FIG. 8

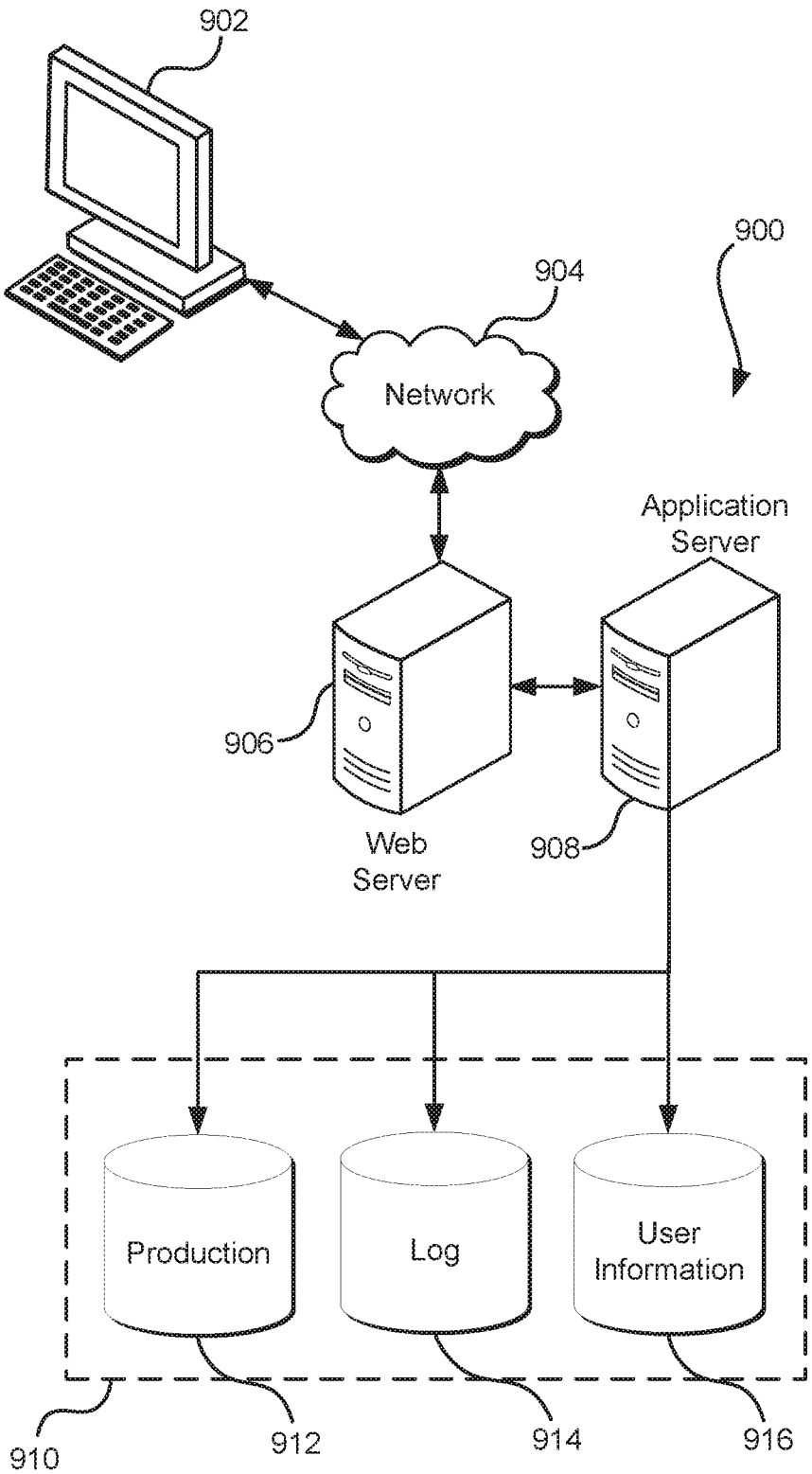

902
900
904
Network
Application Server
906
908
Web Server
Production
Log
User Information
910
912
914
916

FIG. 9

USING CONTAINERS FOR UPDATE DEPLOYMENT

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation of U.S. patent application Ser. No. 14/671,996, filed on Mar. 27, 2015, entitled "USING CONTAINERS FOR UPDATE DEPLOYMENT," the content of which is incorporated by reference for all purposes.

BACKGROUND

Modern computing systems are comprised of a combination of hardware and software, and software providers commonly issue new releases of such software as they find and fix software errors as features of the software are added or removed. Testing the new software releases prior to implementation is critical to ensure that no new problems are introduced to customers; however, it is often difficult to replicate the same usage patterns and hardware, software, and environmental conditions of the live environment within the testing environment. Furthermore, deploying new software releases often require downloading and/or installing an entire software package, even when only minor changes have been made to the software or when the particular device may already have most of the software of the software package already stored on the device. Additionally, even after testing has been performed, when the new software is released to the live environment, there may be a significant period of downtime where affected users and computer systems are unable to perform important tasks while the new releases are being installed and configured within the live environment.

BRIEF DESCRIPTION OF THE DRAWINGS

Various embodiments in accordance with the present disclosure will be described with reference to the drawings, in which:

FIG. **1** illustrates an example of a device with containers in communication with a deployment service in accordance with an embodiment;

FIG. **2** illustrates an example of a deployment service in communication with a deployment agent on a device in accordance with an embodiment;

FIG. **3** illustrates an example of a device with containerization support in accordance with an embodiment;

FIG. **4** illustrates an example of a device with multiple background containers in accordance with an embodiment;

FIG. **5** illustrates an example of testing framework performing testing of a software package for a device in accordance with an embodiment;

FIG. **6** illustrates an example of a testing framework receiving results from testing of a software package on a device in accordance with an embodiment;

FIG. **7** is a flow chart that illustrates an example of deploying a software update to a device in accordance with an embodiment;

FIG. **8** is a flow chart that illustrates an example of testing a software update in accordance with an embodiment; and

FIG. **9** illustrates an environment in which various embodiments can be implemented.

DETAILED DESCRIPTION

In the following description, various embodiments will be described. For purposes of explanation, specific configurations and details are set forth in order to provide a thorough understanding of the embodiments. However, it will also be apparent to one skilled in the art that the embodiments may be practiced without the specific details. Furthermore, well-known features may be omitted or simplified in order not to obscure the embodiment being described.

Techniques described and suggested include a system and method for deploying and testing software packages in software containers on client devices, such as mobile telephones, e-readers, tablet computers, laptop computers, and other computing devices. The system and method include obtaining, at a computing system of a deployment service, a software package comprising a set of computer-executable instructions that have been configured by a software provider to launch and execute within a software container. The deployment service may have a record of client devices that are compatible with the software package, or may have another method of determining to which client devices the software package should be deployed. The deployment service may also have information about the client devices indicating the hardware and indicating which versions of which software may already be installed on a client device. Based at least in part on this information, the deployment service may determine whether the client device has sufficient resources to support installing and/or testing the software package on the client device. Furthermore, based at least in part on this information, the deployment service may also determine which software libraries and other files actually need to be provided to the client device and which software libraries and other files need not be provided because they are already resident on the client device. Note that in some implementations, the deployment service may not make this determination, but rather may provide a list of library files and their versions, and the client device may determine and respond to the deployment service with a list of the library files that it actually needs.

The client device may include hardware and software that supports containerization, and may be configured to have an active software container and at least one background software container. Each software container may have its own separate, isolated user space, but each container may be configured to share resources (such as persistent storage) with another software container as needed. Each software container may be allocated a fixed amount or share of one or more resources, such as processing capability, memory, or storage. The active software container may include the active user space and include the "live" applications usable by the user of the client device. Background containers may contain applications that are actively running or may be idle, but unless made "active," may not be accessible to the user of the device, and the user of the device may not even be aware of the existence of the background container. In some cases, when software is installed to a container, if the software includes certain dependencies, such as specified versions of libraries, that are not present in the software container or within the software package but may be found within another container on the client device or elsewhere on the client device, a container encapsulation engine or a deployment agent running on the client device may retrieve, copy, or share the needed dependency from its current location to complete the software installation in the software container.

The deployment service may poll or query the client devices to determine whether they are ready to receive the software package. When a client device indicates that it is ready to receive the software package, the deployment service may provide the software package to the client

device according to a schedule negotiated with the client device. In some cases, the software package may be received by the client device and installed into a background software container running in parallel with the current active software container on the device, which may be utilized as a test container for the software prior to the software being made live. Installing and testing of the software into the background container may be managed by a deployment agent running on the client device.

If the client device has sufficient resources to test the software in the background container, the deployment agent may cause various tests to be performed on the software. In some situations, the client device may be unable to perform some or all tests on the software; for example, performing some tests, due to the hardware and/or software capabilities of the client device, may unacceptably degrade the performance of the client device for the user during the testing. In such cases, testing may be performed in a test container at a testing framework of the deployment service configured to replicate the conditions and capabilities of the client device. In either case, the results of testing may be logged by the testing framework and used in determining whether testing was successful and whether the software should be made live. In some examples, "live" may refer to a stage in software development where the software has completed testing and is launched and made available to users. Thus, the "live environment" may refer to applications executing in a user space available to the user. In some examples, a "test environment" may refer to applications executing in a user space intended for testing the software prior to the software being transitioned to the live environment. Applications being tested may be complete and testing is being performed to confirm that the software is stable and performs as intended. Another stage may be a "development" stage, and software in a "development environment" may be incomplete and/or otherwise still be in the process of being developed and may not yet be ready to be tested in the test environment.

If the software is determined not to be made live, the test container may be deprovisioned as needed. Otherwise, if the software is determined to me made live, the deployment agent may cause the background container to be switched to be a new active container, and the previously active container may be switched to be a background container. The previously active container may then be deprovisioned as needed or may be retained for a time in case the new active container exhibits a problem and deployment needs to be rolled back. In some cases, the user may be allowed to select between one or more containers on the device as the active container.

The described and suggested techniques improve the field of computing, and specifically the field of software deployment, by, for a computing device targeted to receive a software update, taking advantage of unused or idle resources on the computing device to perform testing of the software update on the computing device in order to closely replicate how the software update will actually perform when made live. Additionally, the described and suggested techniques improve the functioning of computer systems by reducing the resources required to deploy the software update by installing only those portions of the software update not already resident on the computing systems. Moreover, the described and suggested techniques offer meaningful advantages over general software update schemes by reducing the period of downtime required to transition the software update into the live environment.

FIG. **1** illustrates an aspect of an environment **100** in which an embodiment may be practiced. As illustrated in FIG. **1**, the environment **100** may include a device **102** having an active container **104** running a first operating system **106**A and set of applications **108**A in the foreground. The device **102** may also have resources available for one or more test containers, such as the test container **110**, which may receive a second operating system **106**B and/or set of applications **108**B from a deployment service **112**.

The device **102**, in some embodiments, is embodied as a physical device and may be capable to send and/or receive requests, messages, or information over an appropriate network. Examples of such devices include personal computers, cellular telephones, handheld messaging devices, laptop computers, tablet computing devices, set-top boxes, personal data assistants, embedded computer systems, electronic book readers, and the like, such as the electronic client device **902** described in conjunction with FIG. **9**. Components used for such a device can depend at least in part upon the type of network and/or environment selected. Protocols and components for communicating via such a network are well known and will not be discussed in detail. Communication over the network can be enabled by wired or wireless connections and combinations thereof.

The device **102** may be configured to support containerization; that is, the execution of applications and/or operating systems inside software containers. A container (also referred to as a software container or isolated user space instance) may be a lightweight operating system-level virtualization environment/instance running under a computer system instance that includes programs, data, and system libraries. When the software container is run, the running programs/processes in the software container may be isolated from other processes running on the same computer system. Thus, the active container **104** and the test container **110** may each run under a container encapsulation system. In some examples, a container encapsulation system may allow one or more containers to run within a single operating instance without the overhead associated with starting and maintaining virtual machines for running separate user space instances. In some examples, "user space" may refer to memory logically separated and allocated to a set of applications such that processes executing one user space cannot, unless explicitly allowed, access memory of applications executing in another user space, and vice versa. An example of a container encapsulation system is the Docker container engine. Operating systems **106**A-**06**B of the software container instances **104** and **110**, respectively, may execute in isolation from each other (e.g., each container may have in isolated view of the file system of its respective the operating system). Each of the software containers **104** and **110** may have their own namespace, and applications running within the software containers **104** and **110** may be isolated by only having access to resources available within the software container namespace. Thus, the software containers **104** and **110** may be an effective way to run one or more single applications within their own namespace.

The active container **104** (i.e., the foreground container) may be a container holding an operating system and applications currently in use by a user of the device **102**. That is, when the user uses the device **102** and chooses to execute an application, said application may be running under the first operating system **106**A and selected from one of the set of applications **108**A. Conversely, the test container **110** may be a container executing in the background. In some cases, the test container **110** may be executing in the background of the device **102** such that the user is not provided any

indication that the test container **110** is executing on the device **102**. In some embodiments, there may be multiple test containers on the device, with each test container executing a different operating system and/or set of applications. The test container **110** may be configured to receive software, such as updates to the version(s) of software currently installed in the active container **104**, from the deployment service **112**. Such software or software updates may be a set of files created by a provider of software for the device **102** and may be provided to the deployment service **112** for deployment to the device **102**.

The test container may be further configured to perform tests on the received software, such as executing a script to determine whether the received software has any software errors or defects or has any issues (e.g., compatibility) with the device **102**. For example, such testing may evaluate whether the received software executes within the test container **110**, responds correctly to various inputs, performs its functions within an acceptable time, is stable, and achieves the intended results. Test results may be uploaded to the deployment service **112** for further analysis. In some embodiments, use of resources (e.g., processor, memory, persistent storage, etc.) by an active user space, such as the user space of the active container **104**, may take priority over testing that involves those resources, such that testing that involves those resources may be delayed until the device **102** is idle (e.g., no user input for 15 minutes, processor usage drops below 20%, available memory exceeds 75% of total, etc.) or until such resource is no longer in active use by another user space. In some cases, if, after a predetermined interval, testing is unable to be performed because resources needed for testing are in use, a testing framework of the deployment service **112** may be utilized to perform testing, as described in the present disclosure, in lieu of performing testing on the device itself.

The first operating system **106**A may be an operating system, such as Linux, Microsoft Windows, or Google Android, configured to run on the device **102**. The set of applications **108**A may similarly be one or more applications configured to run under the environment of the first operating system. Such set of applications **108**A may include applications such as e-mail clients, word processing applications, Internet browsers, and multimedia applications.

Likewise, the second operating system **106**B may be the same as or different from the first operating system **106**A. For example, the second operating system **106**B may include an update (e.g., patch, new version, etc.) to the first operating system **106**A. Alternatively, the second operating system **106**B may be a completely different operating system than the first operating system **106**A; for example, the first operating system **106**A may be a Linux operating system, while the second operating system **106**B may be a Microsoft Window operating system. Similarly, the set of applications **108**B may be the same as or different from the set of applications **108**A. As an example, the second operating system **106**B may be the same as the first operating system **106**A, however one or more of the set of applications **108**B may be different from the set of applications **108**A (e.g., new application version, one or more additional applications, one or more application removed, etc.). The second operating system **106**B and applications **108**B may be received from the deployment service **112** in a manner that minimizes impact to the user of the device **102**. For example, the deployment service **112** may notify the device **102** that an update is available.

The device **102** may respond to the deployment service **112** when it is ready to receive the software update (e.g., when the device is not actively in use by the user; idle). If the device **102** does not indicate that it is ready to receive the software update, the deployment service **112** may wait for a certain interval and query the device **102** again whether it is now ready to receive the software update. Not only may the deployment service **112** query whether the device **102** is ready to receive the update, the deployment service **112** may also query whether the device **102** has sufficient resources (e.g., storage space, network bandwidth, specified processing capacity, sufficient available memory, etc.) to receive the software update. If the device **102** indicates that it has insufficient resources to receive the update, the deployment service **112** may delay providing the software update to the device **102** until such time as the device **102** indicates that it has sufficient resources available (e.g., deleted/uninstalled unused files to free up storage space, upgraded memory, etc.).

When the device **102** indicates to the deployment service **112** that it is ready to receive the software update, the deployment service **112** may provide the software update, such as through the Internet, WiFi, or other network connection, to the device. In some implementations, the device **102** may be configured with a preference to receive software updates during times when the device **102** is not being actively used by a user of the device. In such implementations, should the user attempt to use the device **102** while the device is in the midst of receiving a software update, the device **102** may notify the deployment service **112** to suspend transmission of the software update until such time that the device **102** notifies the deployment service **112** that it is once again ready to receive the software update. In some cases, receipt of the software update may require consent by the owner of the device **102**, and consent may be granted through an interface, such as a confirmation dialog, on the device **102**.

If the second operating system **106**B and applications **108**B are determined to be fully compatible (e.g., based at least in part on the results of testing the software update) and ready for use by the user of the device **102**, the device **102** may be instructed to change the status of the test container **110** to be the active container. Similarly, the active container **104** may be rendered inactive. In some embodiments, the inactivated active container **104** may be retained for a period of time in case the inactivated active container **104** needs to be made active again (e.g., if the second operating system **106**B and/or applications **108**B are later found to have defects or other issues). In this manner, software updates may be provided to the device **102** with little impact on the user of the device **102**.

The deployment service **112** may be a service provided by a computing resource service provider for deploying software updates to computing devices, such as the device **102**. For example, a provider of cellular telephone software may be a customer of the computing resource service provider and may its upload software updates to the deployment service **112**, such as through an interface (e.g., application programming interface) made available to the customer by the computing resource service provider. The software updates may be encapsulated as a containerized set of computer-executable instructions (also referred to as a "software package"), which may represent an entire copy of the particular state of the software container at the time the software package was generated. The software package may be configured to launch and execute as a software container after the software package obtained by the device (e.g. after the software package is provided to the test container **110** of the device **102** by the deployment service **112**).

The deployment service **112** may have information that identifies appropriate devices, such as the device **102**, for which the uploaded software updates are intended (e.g., records of media access control addresses of devices having older versions of the software). In some cases, the device **102** may be configured to, once it has power and a network connection sufficient to communicate with the deployment service, identify itself to the deployment service **112** through the network connection. The device **102** may report its hardware configuration and/or software (e.g., libraries, drivers, etc.) and software versions that are installed on the device **102** to the deployment service **112**. The deployment service **112** may store this information in a data store for reference in future determinations whether the device **102** is in need of a software update or not. In some implementations, a manufacturer or vendor of the device **102** may provide this information to the deployment service **112**. Similarly, this information may be updated as software updates are installed or rolled back on the device **102**. Based at least in part on information about the hardware and currently installed software on the device, the deployment service **112** consequently may determine, for each such device, whether the device is capable of supporting a test container, and, if so, pushing the received software update to the test container **110**. That is, the deployment service **112** may schedule with the device **102** how and when to provide the software update to the test container **110** on the device **102**. For example, the software update may be scheduled to be provided immediately to the device **102**, may be scheduled to be provided to the device **102** at a specified date/time, or may be scheduled to be provided to the device **102** when the device **102** indicates that certain conditions have been met (e.g., device has been idle for 30 minutes). As another example, the software update may be provided to the device **102** in multiple portions, and the deployment service **112** may be scheduled to provide each portion to the device **102** in a certain order or at certain times. Once the software updates are launched in the test container **110**, such software updates may undergo testing or may be made active as needed.

The active container **104** and the test container **110** may be launched to have only specified resources from resources allocated to the respective container instance; that is, a container may be launched with a specified allocation of memory and may be specified to not utilize more than a certain amount of processing power. As noted, multiple containers may run simultaneously on a single device, and the resources of the device can thus be distributed between the software containers as needed.

FIG. **2** illustrates an environment **200** in which an embodiment may be practiced. As illustrated in FIG. **2**, the environment **200** may include an owner **224** of a device **202** having a deployment agent **214** executing on the device **202**. The deployment agent **214** may be configured to act as an intermediary between the device **202** and the deployment service **212**. The deployment service **212** may comprise one or more functional modules, supported by hardware and software, for receiving software updates from customers **226** of the deployment service **212** and for providing said software updates to devices for which the software updates are intended, such as the device **202**. The functional modules may include a deployment engine **216** configured to receive software packages from the customers **226** and schedule deployment of the software packages with devices, as well as a cache fleet **220** and/or private hubs **222** configured to cache software packages and/or component files for deploying the software packages to the devices.

The owner **224** may be an authorized user of the device **202**. For example, the owner **224** may be the owner (or other authorized possessor) authorized to confirm the installation of software updates to the device **202**. The device **202** may be computing device, such as the computing devices described in conjunction with the device **102** of FIG. **1**. As noted, the customers **226** may be providers of software for devices such as the device **202**. Examples of such customers include cellular telephone providers, operating system distributors, application developers, and the like. Note that the entity represented by the customers **226** may include the computing resource service provider that provides the deployment service **212**, should the computing resource service provider also provide software packages for devices like the device **202**.

As noted, the deployment service **212** may be a service made available to the customers **226** and device owners for deploying software packages and software updates to devices like the device **202**. The deployment service **212** may provide interfaces for communicating software packages to the deployment service **212** as well as interfaces for delivering the software packages to devices. The deployment service **212** may be implemented as one or more physical and/or virtual computing devices within a distributed computing environment, and may be configured to communicate to the customers **226** and to devices via a network, such as the Internet.

The device **202** may be configured to include a deployment agent **214**. The deployment agent **214** may be integrated into the hardware of the device **202** or may be implemented as a software application, and may be configured to manage software deployment on the device **202**. For example, the deployment agent **214** may be configured to communicate with and receive update information from the deployment engine **216**, the cache fleet **220**, and/or the private hubs **222**. For example, the deployment engine **216** may notify the deployment agent **214** on the device **202** that a software update is available. The deployment engine **216** may also query the deployment agent **214** whether the device is available to receive the software update, whether the device has sufficient resources to receive the software update, and whether the device has sufficient resources to test the software update. The deployment agent **214** may be configured to respond accordingly to the queries of the deployment engine **216**. If the device **202** has sufficient resources to run at least some tests on the software update, the deployment agent **214** may also be configured to cause tests to be run on software running in test containers and may provide results of the testing to the testing framework **218**.

The deployment engine **216** may be comprised of one or more computing systems and may be configured to receive software packages from customers **226** of the deployment service **212**, such as through an application programming interface. The deployment engine **216** may be further configured to receive values for one or more parameters of the received software packages, such as hardware and/or software requirements for executing the software packages. After receiving a software package, the deployment engine **216** may be configured to notify the device **202** that the software package is available.

In response to the notification of a software package from the deployment engine **216**, the deployment agent **214** may first determine whether the device **202** has sufficient resources to launch the software package in a separate container. If the device **202** has insufficient resources to launch the software package in a separate container, the

deployment agent **214** may communicate this detail to the deployment engine **216**, whereupon the deployment engine **216** may determine whether to provide the software package to the testing framework **218** for testing instead of the device **202**. Otherwise, if the device **202** has sufficient resources to launch the software update in a separate container, the deployment agent **214** may cause a new container to be instantiated in the background of the device **202**, similar to the test container **110** of FIG. **1**, if a suitable container is not already instantiated on the device **202**. The deployment agent **214** may also be configured to query the deployment engine **216** about the software package contents (e.g., software libraries and other dependencies) and/or provide the deployment engine **216** with information about current versions of software libraries (e.g., glibc-2.21) or other dependencies on the device **202**.

Using this information, the deployment agent **214** or the deployment engine **216**, depending on the implementation, may determine differences between the software package and software currently resident on the device **202**. In this manner, one software package may be created for multiple devices having different configurations, and only portions of the software package different from the software currently resident on individual devices need be provided to the individual device by the deployment service **212**. Bandwidth usage and other resource usage may be conserved thereby because redundant software libraries and other dependencies need not be downloaded from the deployment service **212** when they are already resident on the device **202**.

The cache fleet **220** may be one or more physical or virtual computing systems in a distributed computing environment. The cache fleet **220** may be configured to cache/store software libraries and/or software packages for deployment to compatible devices, such as the device **202**. As such, one or more computing systems of the cache fleets may be located in various geographic regions, such that the software libraries and/or software packages can be provided quickly and efficiently to the device **202** from a computing system in a geographic region near to the device **202**. The computing systems of the cache fleets may be configured to communicate the software libraries and/or software packages in parallel to multiple devices similar to the device **202**. The deployment agent **214** or the deployment engine **216**, depending on the embodiment, may communicate to the cache fleet **220** which software libraries and/or other dependencies are needed by the device **202** for the software package, based on differences between the software in the software package and software currently resident on the device **202**. In response, the cache fleet **220** may provide the identified software libraries and/or other dependencies to the device **202** as needed.

Note, in some embodiments, the deployment service **212** may implement private hubs **222** alternative to or in addition to the cache fleet **220**. The private hubs **222** may operate similar to the cache fleet **220** (i.e., keeping caches of software libraries, other dependencies, and/or software packages, located in various geographic regions, etc.), but each cache hub may be assigned to a specific type of device or customer. That is, the private hubs **222** may contain cache files specific to certain device configurations. Unlike the cache fleets **220**, an individual private hub of the private hubs **222** may be secured against access by users, customers, and devices other those authorized to access the private hub. For example, cellular telephone Carrier A may be allocated a private hub to cache software packages customized for use by particular models of cellular telephones supported by Carrier A. Similarly, cellular telephone Carrier B may be allocated a different private hub for software packages customized for use by particular models of cellular telephones supported by Carrier B. Software packages on Carrier A's private hub may be unavailable to the users cellular telephones of Carrier B, and vice versa. The private hubs **222** may be geographically located to provide efficient software delivery to the devices targeted; for example, if Carrier A primarily supports devices in geographic region X, the private hubs **222** allocated to Carrier A may be located within geographic region X to enable devices in geographic region X to download software packages more efficiently. Furthermore, in some embodiments commonly-available software libraries may still be available to be downloaded from the cache fleet **220**; in this manner, the private hubs **222** may minimize storage requirements by only hosting customized portions of software packages. Therefore, each of the private hubs **222** may be able to deliver efficiently, to its corresponding device, the files necessary to update the particular device.

As noted in the discussion of FIG. **1**, the software packages may be delivered to a background container on the device **202**. The running processes and resources within the software container may be isolated from other processes and resources on the device **202** and the background container. The amount of memory and processing capacity to be allocated to the background container may be specified by the customer **226** that created the software package. The customer may create the software package by configuring a software container and configuring applications and data to execute within the software container. The customer **226** may then package the configured container, applications, and data as an image (e.g., the software package), including one or more programs, data, and any system dependencies (e.g., libraries, files, etc.) that are needed for the programs to run on the operating system of the container instance. In some examples, a software "package" may refer to an entire copy of a particular state of the software container at the time the image was generated. The software package thereafter may be used to launch one or more identical containers, each of which may be assigned the specified amount of resources and may be isolated from each other. In this manner, the customer **226** can generate a software package for deployment to multiple devices. The software packages may be launched within containers on devices having the same or different hardware, and each container may be expected to run in a similar manner as the originally packaged software container.

Note that the present disclosure may refer to both a software package, in reference to software packaged for delivery to a device, as well as software "updates," indicating a software package that is different (e.g., newer version, patch containing bug fixes, etc.) from software currently resident on the device **202**. However, it is contemplated that, unless otherwise indicated, functionality described with reference to software updates may also be applied software packages, including software that has not yet been installed on the device **202**. Note also that one or more of the deployment engine **216**, the cache fleet **220**, the private hubs **222**, and the testing framework **218** may not be present in various embodiments of the deployment service **212**. For example, the various functionalities described in reference to the deployment engine **216**, the cache fleet **220**, and/or the private hubs **222** may be integrated into other components of the deployment service **212**, or may be separated into additional or different modules than depicted in FIG. **2**.

The testing framework **218** may be involved in different aspects of testing of software updates before the software

updates are made "live" (e.g., by the software container in which the software updates are installed being switched to be the active container). A first aspect of the testing framework **218** may include receiving test results from the device **202**. In some cases, this aspect may also include analyzing the test results and determining whether the software update should be made live based at least in part on whether the test results indicate that testing was successful or not. Note that in some cases, a software update may not be made live even if testing was successful. For example, a software provider may intend only to collect information about how a particular software package or update would behave on the device **202**, but may not intend to release the particular software package or update to the public. In such a case, the software package or update may be tested in a container in accordance with the present disclosure, but the software container may never be made a "live" container. In these cases, the test container may be deprovisioned after testing is complete and the test results are received by the testing framework **218**. In some examples, "deprovisioning" may refer to the act of removing access to a software container and freeing up resources allocated to the software container. Deprovisioning a software container may include stopping the container from running, returning any resources (e.g., memory, processing capacity, etc.) allocated to the software container to be available to other processes of the device, and removing any image or other files dedicated to the software container from persistent storage. Reasons for such cases include situations in which the customer **226** (e.g., software provider) may intend to use such collected information for future software development purposes and situations in which the customer **226** (i.e., software provider) only wants to test the software on a limited number of devices before rolling the software out, live, to a wider audience.

A second aspect of the testing framework **218** may include providing an alternative platform for testing the software updates if the device **202** has insufficient resources or is otherwise unable to perform the testing itself (e.g., the device **202** is offline, the device is busy, the owner has not given consent for the software update to be installed and/or tested on the device **202**, etc.). That is, the testing framework **218** may include one or more physical or virtual computing systems configured to simulate the hardware and/or software of the device **202**. For example, the testing framework **218** may cause the software update to be installed in a container of a virtual machine instance configured to simulate the device **202**. The testing framework **218** may further cause tests to be performed on the software update in the virtual machine instance as if the tests were being performed on the device **202** itself, and the testing framework **218** may further receive the results of the testing at least in part for determining whether the software update should be made live. In some embodiments, the software for the virtual machine instances may be obtained from the deployment engine **216**. On other embodiments, the software for the virtual machine instances may be obtained from the cache fleet **220** or the private hubs **222**.

In some implementations where testing of the software package was performed on the device **202**, the deployment agent **214** may be configured to determine whether testing was successful. In other implementations, the deployment agent **214** may communicate the results of the testing to the testing framework **218** (e.g., may provide a stack trace of the testing to the testing framework **218**), and the testing framework **218** may determine whether testing was successful. In still other implementations, the deployment agent may communicate the results of the testing to the deployment engine **216** or may communicate the results of the testing to the testing framework **218** which may be in communication with the deployment engine **216**, and the deployment engine **216** may determine whether testing was successful. In some implementations where testing of the software package was performed by the testing framework **218**, the testing framework **218** may determine, based at least in part on the results of the testing, whether testing was successful. In other implementations, the testing framework **218** may communicate the results of the testing to the deployment engine **216** and the deployment engine may determine whether testing was successful.

Upon determination of successful testing, the deployment engine **216** may query the deployment agent **214** whether the device **202** is ready to accept the software package in an active container. As noted, in some cases, the device **202** may have sufficient resources to install the software package in a background container and perform some or all testing at the device **202** itself. In other cases, the device **202** may have sufficient resources to install the software package in a background container, but some or all testing may be performed at the testing framework **218** in an environment that simulates the device **202**. Upon acceptance by the deployment agent **214** (which may, in some implementations, prompt the owner **224** for consent to accept the software package and may not indicate acceptance of the software package until such consent is received), if the software package is already installed on the device **202** in a background/test container, the deployment agent **214** may cause the background/test container to be switched to become the active container, and cause the previously active container to be switched to be an inactive container or switched to run as a background container. In some cases, the deployment agent **214** may cause this switching to occur in response to a an instruction to do so from the deployment engine **216**. In some of these cases, the deployment agent **214** may send this instruction in response to a command to do so from the customer **226** that provided the software package (e.g., testing may have been successful, but the software provider may prefer delay making the software live until a certain date, etc.).

In still other cases, the device **202** may have insufficient resources to perform testing of the software package on the device **202** itself, or may have insufficient resources to install the software in a background container on the device **202**, and in such cases, testing may be performed at the testing framework **218** as described in the present disclosure. In these cases, upon acceptance by the deployment agent **214** (which, as noted, may require obtaining consent from the owner **224**), the software package may, if resources of the device **202** permit, be downloaded and installed in a background container, and the background container switched to be the active container instead of the previously active container, or, if resources on the device do not permit installation in a background container, be downloaded and installed to the active container of the device **202**. Note, that if results of testing are determined to be unsuccessful, the test container and/or software package, whether located on the device **202** or the testing framework **218**, may be deprovisioned from the entity upon which it was provisioned/instantiated.

FIG. **3** illustrates an example embodiment **300** of the present disclosure. Specifically, FIG. **3** depicts a device **302** with system hardware **328** that supports running a container encapsulation engine **330**. The container encapsulation engine **330** may support running one or more containers, such as an active container **304**. The container encapsulation

engine **330** may also support running one or more background containers, such as a test container **310**. The software containers may be configured with respective operating systems **306**A-**06**B and one or more applications **308**A-**08**B. The container encapsulation engine **330** may further be configured to run a deployment agent **314** for communicating with a deployment service and for managing software deployments on the device **302**. The device **302** may be a device similar to the types of devices described for the device **102** of FIG. **1**. The active container **304** may be a software container configured run in the foreground, similar to the active container **104** of FIG. **1**, such that the user is able to interact with the first operating system **306**A and applications **308**A executing within the active container **304**.

The test container **310** may be configured to run in the background and may contain the second operating system **306**B and applications **308**B, similar to the test container **110** of FIG. **1**, which may not interact with the user on the device. Rather than interact with the user, the test container **310** may be inactive unless it is provided input from the deployment agent **314**. For example, the deployment agent **314** may receive instructions for testing the second operating system **306**B and/or applications **308**B from a deployment service. In this example, in accordance with the testing instructions, the deployment agent **314** may provide simulated user input to the second operating system **306**B and/or applications **308**B in order to monitor how the second operating system **306**B and/or applications **308**B respond to the provided input. The test container **310** may be one of multiple test containers. Each such test container may be installed with the same software package, but may be caused to perform the same or different tests on the software in parallel with the other test containers. For example, a first test container may be configured to perform boundary testing on the software in the first test container, while a second test container may be configured to perform fault injection testing on the software in the second test container, and a third test container may be configured to perform battery life testing in the third test container.

Additionally or alternatively, multiple test containers may be installed with different software packages, and each test container may be caused to perform tests on its installed software package in parallel with other test containers. In some implementations, the operating system **306**B and/or one or more of applications **308**B may be instrumented to measure the execution performance of the software, trap errors, and/or provide trace information to the deployment agent **314**. The deployment agent **314** may, in turn, communicate the provided information to a testing framework of the deployment service for determination of the level of success of the testing.

The operating systems **306**A-**06**B may be configured to execute as operating systems in isolation from each other. The first operating system **306**A may be the same or different operating system as the second operating system **306**B. Likewise, the applications **308**A-**08**B may be configured to execute within their corresponding operating systems **306**A-**06**B, and the first set of applications **308**A may be the same or different as the second set of applications **308**B.

The deployment agent **314** may be an application running on the device **302** that is configured to communicate with a deployment service. The deployment agent **314** may be configured to respond to queries by the deployment service, such as queries as to whether the device **302** is available to accept a software package, has sufficient resources available to receive the software package, and has sufficient resources to test the software package. The deployment agent **314** may also be configured to receive results from testing of the installed software packages, such as the second operating system **306**B and/or **308**B, within the test container **310**, and may communicate those results to the deployment service. Note that the deployment agent may be configured to run in various ways, including within one or more containers, within its own container, or to run in parallel with the container encapsulation engine **330**. Furthermore, in some implementations, there may be multiple deployment agents on the device **302**; for example, a device **302** may have multiple test containers for testing different software packages, and each test container may have a corresponding deployment agent.

The system hardware **328** represents the hardware of the device **302** implemented to support the other components of the device **302** depicted in FIG. **3**, including memory, one or more processors, firmware, input/output devices, display devices, and other hardware as may be necessary to support containerization. The container encapsulation engine **330** may be a software system executing on the system hardware **328** and configured to allow one or more software containers, such as the active container **304** and test container **310**, to run within a single operating instance. Examples of container encapsulation systems include the Docker container engine, the Parallels Virtuozzo container engine, and the Solaris container engine.

The example embodiment **300** further illustrates a blue-green deployment strategy that may be implemented by embodiments of the present disclosure. Blue-green deployment minimizes downtime necessary to take software from a testing stage to live production. Blue-green deployment accomplishes this downtime reduction by performing its testing in a test environment (green environment) as identical to the live production environment (blue environment) as possible. Once testing of the green environment indicates that the software is ready to go live, the green environment is switched to become the live environment, and the blue environment is allowed to become idle. Thus, as illustrated in FIG. **3**, when the test container **310** is determined to be ready to go live, the status of the test container **310** (green environment) may be switched to become a new active container, such that user input and output flow to and from the new active container. Similarly, the status of the previously active container **304** may be changed such that the previously active container **304** becomes inactive/idle. Another advantage of blue-green deployment is that if any issues arise, the blue environment can be quickly reactivated and the green environment can likewise be deactivated. Thus, in the present disclosure, if issues are discovered with the new active container, the new active container can be deactivated and the previously active container **304** may be reactivated as the active container.

FIG. **4** illustrates an aspect of another environment **400** in which an embodiment may be practiced. Specifically, FIG. **4** depicts a device **402** having an active container and a plurality of background containers **410**. A deployment service **412** may provide software packages through a network **432** via a container agent **414** acting as an intermediary between the background containers **410** and the deployment service. The deployment service **412** may be a service similar to the deployment service **212** of FIG. **2**. The device **402** may be a device like the device **102** or the device **202** described in conjunction with FIGS. **1** and **2** respectively. Likewise, the active container **404** may be a container similar to the active container **104** of FIG. **1** or the active container **304** of FIG. **3**.

The network **432** represents the path of communication between the deployment service **412** and the device **402**. Examples of the network **106** include the Internet, a local area network, a wide area network and Wi-Fi. The background containers **410** may be configured to launch and test one or more software packages received from the deployment service **412**. Alternatively or additionally, one or more of the background containers **410** may contain a software configuration that may be utilized as an alternate active container, and the user of the device **402** may select which of the background containers **410** the user wishes to be the current active container **404**. Alternatively or additionally, a provider of software packages for the device (e.g., cellular telephone carrier) may select which of the background containers **410** to use on the device as the current active container **404**. For example, the active container **404** may include a first operating system and a first set of applications compatible with the first operating system. A background container of the plurality of background containers may include a second operating system, different from the first operating system, and a second set of applications compatible with the second operating system. In this manner, the user may be able to cause the device to switch the background container to be the active container in order to use an application from the second set of applications, such as in the event that the application from the second set of applications is incompatible with the first operating system. There may be any number of such alternate containers on the device **402**, only as limited by the resources available to the device **402**.

FIG. **5** illustrates an aspect of an environment **500** in which an embodiment may be practiced. Specifically, FIG. **5** depicts a device **502**, having insufficient resources to test a software package **506** interacting with a testing framework **518** and a deployment engine **516** of a deployment service **512**. The device **502** may be a device like the device **102** or the device **202** described in conjunction with FIGS. **1** and **2** respectively. The device **502** depicted in the environment **500**, as noted, may be a device capable of running the software of the software package **506**, but may be unable to perform adequate testing of the software. For example, one or more of the intended tests may require more time or resources to perform than would be acceptable by a user of the device **502**. As another example, a user of the device **502** may have denied consent to allow such testing to be performed on the device **502** (e.g., through settings on the device or through input on a confirmation dialog). As yet another example, the device **502** may be temporarily out of communication with the deployment service **512** (e.g., the device may be outside of WiFi range) and therefore unable to receive the software package **506** for testing. Note that in some cases, there may be some tests that may be performed on the device **502** itself, as described in the present disclosure, parallel with other tests being in parallel in the test container **510** of the testing framework **518**. For example, the testing framework **518** may be configured to perform more resource-intensive tests than those being performed on the device **502** itself.

As noted, the deployment service **512** may be a service provided by a computing resource service provider for deploying software packages, such as the software package **506**, to computing devices, such as the device **502**. The software package **506** may be a software package or software update, as described in the present disclosure, configured to launch as software in a software container. The contents of the software package **506** may vary, and may include, for example, configuration file data, database records, one or more software applications, library files, precompiled executable instructions, uncompiled software code/executable instructions, and/or entire operating systems. The software package may be provided to the deployment service by a software provider and may be configured to be used only for specific devices, in this case, the device **502**.

The test container **510** may be a software container intended to simulate an active container running on the device **502**, and, as such, may be implemented as may be a virtual machine instance or may be configured to run within a virtual machine instance of the testing framework **518**. Thus, the software package **506** may be launched within the test container **510**, just as it would be launched in a software container on the device **502**. Like testing performed on devices as described in the present disclosure, each application within the test container **510** may need to be tested; for example, even if only one application of the software package **506** is different or updated from the applications actively running on the device **502**, every other application within software package **506** may need to be tested in the test container **510** to confirm that the application that changed does not cause issues with applications that have not changed. Test data, such as errors, completion times, warnings, and successful test results may be logged by the testing framework **518** for later determination whether the testing was successful, and may also be provided to the software provider of the software package **506** to aid future software development and/or determining the cause of software errors.

The deployment engine **516** may be comprised of one or more computing systems and may be configured to interact with the device **502**, including notifying the device **502** that testing for the software package **506** was successful and that the device **502** may, if not already downloaded, download the software package **506** and launch it in a container on the device **502**. Once launched on the device **502**, such notification may also serve as notice to the device **502** that it may, based at least in part on the successful test results, safely make the software container containing the software package **506** on the device **502** its active container.

The testing framework **518** may test containers, such as the test container **510** corresponding to the device **502**. The testing framework **518** may obtain the software package **506** intended for the device **502** from the caches **520**, install the software package **506** in the test container **510**, and cause tests to be performed on the software in the test container **510** as if the tests were being performed on the device **502** itself. The testing framework **518** may also determine, based at least in part on the test results, whether the software package **506** should be made live on the device **502**.

The caches **520** may be one or more physical or virtual computing systems in a distributed computing environment, similar to the cache fleet **220** and or the private hubs **222** of FIG. **2**. The caches **520** may be configured to store software libraries or software packages (including software updates) for deployment to devices, such as the device **502**.

FIG. **6** illustrates an aspect of an environment **600** in which an embodiment may be practiced. Specifically, FIG. **6** depicts a device **602**, having sufficient resources to test a software package **606** interacting with a testing framework **618** and deployment engine **616** of a deployment service **612**, and providing test results **636** to a data store **634** of the deployment service **612**. The device **602** may be a device like the device **102** or the device **202** described in conjunction with FIGS. **1** and **2**, respectively. The device **602** depicted in the environment **600**, as noted, may be a device

capable of running the software of the software package **606** and also capable of performing some or all testing of the software. As noted, the deployment service **612** may be a service provided by a computing resource service provider for deploying software packages, such as the software package **606**, to computing devices, such as the device **602**.

The testing framework **618** may be configured to receive the test results **636** of the testing performed on the software package **606** provided to the device **602**, and may be configured to store the test results **636** in the data store **634**. The data store **634** may be a database, set of files, or other storage device or component for storing data. The testing framework **618** may also determine, based at least in part on the test results **636**, whether the software package **606** should be made live on the device **602**. The caches **620** may be one or more physical or virtual computing systems in a distributed computing environment, similar to the cache fleet **220** and/or the private hubs **222** of FIG. **2**. The caches **620** may be configured to store software libraries, software packages (including software updates) for deployment to devices, such as the device **602**.

The software package **606** may be a software package or software update, as described in the present disclosure, similar to the software package **506** described in conjunction with FIG. **5**, configured to launch as software in a software container. The software package may be provided to the deployment service by a software provider and may be configured to be used only for specific devices, in this case, the device **602**. The deployment engine **616** may be comprised of one or more computing systems and may be configured to interact with the device **602**, including notifying the device **602** that testing for the software package **606** was determined by the testing framework to be successful, and that the device **602** may safely make its test container its active container.

The software package **606** may be launched within a test container on the device **602**, similar to the test container **110** on the device **102** of FIG. **1**. Each application of the software package **606** may need to be tested within a test container; for example, even if only one application of the software package **606** is different or updated from the applications currently running in an active container on the device **602**, every other application within software package **606** may need to be tested to confirm that the application that changed does not cause issues with applications that have not changed. Test data, such as errors, completion times, warnings, and successful test results may be provided by the device **602** as the test results **636** to the data store **634** of the testing framework **618** for later determination whether the testing was successful or for providing to the software provider of the software package **606** to aid future software development and/or determining the cause of software errors.

A deployment agent running on the device **602** may be configured to determine a balance between resource usage on the device **602** and test performance for testing. For example, in a situation in which network bandwidth is a constraint and processing capability is in abundance, processing resources may be utilized to compress data streams being communicated through the network. Conversely, in situations in which there is an excess of network bandwidth but processing power is limited, the deployment agent may be configured to cause fewer processing resources of the device **602** to be used for testing and cause some of the processor-intensive testing to be performed at the testing framework, such as in the manner described in FIG. **5**.

FIG. **7** is a flow chart illustrating an example of a process **700** for deploying a software update to a device in accordance with various embodiments. The process **700** may be performed by any suitable system, such as a server in a data center, multiple computing devices in a distributed system of a computing resource service provider, or any electronic client device such as the electronic client device **902** described in conjunction with FIG. **9**. The process **700** includes a series of operations wherein a customer of a deployment service pushes a software update to the deployment service, and the deployment service sends the update to devices that indicate they are available to receive the software update.

In **702**, a provider of software for certain computing devices may create a containerized software package for deployment to the certain computing devices. As noted, the software package may include a set of one or more files configured to execute within a software container on the device. The software package may comprise new software for specified devices or may comprise updates (e.g., bug fixes, additions, or removals to existing software, etc.) to software currently resident on specified devices. The software package may comprise at least a portion of the files and libraries necessary to execute the packaged software, although, as will be described, in many cases, the software package may include all files necessary to execute the packaged software and the deployment service or a deployment agent on the device and/or the container encapsulation engine on the device may selectively choose which files of the set of files need to be downloaded to the device. That is, files and libraries resident on the device of the same version as in the software package need not be downloaded to the device, as the device already has them, thereby conserving network and storage resources.

In **704**, the software provider may provide the software package to a deployment engine of a deployment service, such as via an application programming interface provided by the computing resource service provider that provides the deployment service.

In **706**, the deployment service may notify a device identified as a potential recipient of the software package that the software is available and/or may query the device as to whether the device is ready to receive the software package.

In **708**, a determination is made whether the device has indicated that it is ready to receive the software package. For example, if the device indicates that it is not yet ready to receive the update (e.g., the device is busy, authorized user has not given consent, etc.), the system performing the process **700** may return to **706** (possibly after waiting a predetermined interval) and poll the device again. Similarly, if no response is received from the query of **706**, the system performing the process **700** may wait a predetermined interval and re-poll the device in **706**. Otherwise, if acknowledgement is received from the device that the device is ready to receive the software package, the system performing the process **700** may proceed to **710**, whereupon a determination may be made as to whether the device has sufficient resources to receive the software package.

If the device has insufficient resources to receive the software package, the system performing the process **700** may proceed to **712** where it may log this detail and/or respond to the device that the software package cannot be provided. For example, the device may have insufficient storage available to receive the software package, and therefore the device may need to free up some storage space (e.g., by deprovisioning idle containers, deleting unnecessary

files, etc.). As another example, the device may have insufficient random access memory or processing capability available, and therefore the device may not be able to receive the software package until the device is updated to meet certain minimum requirements.

Otherwise, in **714**, if the device has sufficient resources to receive the software package, the system performing the process **700** may provide the software package to the device, such as through an Internet or WiFi network connection.

Lastly, in **716**, the system performing the process **700** may cause the software update to be deployed in a container on the device. In some implementations, this may be caused by a deployment agent on the device launching the software package within a container on the device. As noted in the present disclosure, in some cases, the software package may be launched into a test container, whereupon a series of tests may be performed on the software package before the test container is switched to be the active container.

Note that one or more of the operations performed in **702-16** may be performed in various orders and combinations, including in parallel. For example, in some implementations, the system performing the process **700** may determine whether the device has sufficient resources in **710** prior to determining whether the device is ready for updates in **708**.

FIG. **8** is a flow chart illustrating an example of a process **800** for testing a containerized software package in accordance with various embodiments. The process **800** may be performed by any suitable system, such as a server in a data center, multiple computing devices in a distributed system of a computing resource service provider, or any electronic client device such as the electronic client device **902** described in conjunction with FIG. **9** in combination with a suitable device, such as the device **102** of FIG. **1**. The process **800** includes a series of operations wherein a device receives a software package, the software package is installed and tested, and, based on the results of the testing, the software package is made live.

In **802**, a software package is received by a device in a manner that may be similar to the operations of **716** of FIG. **7**. That is, the software package may be a set of files created by a provider of software for the device and provided to a deployment service, and the device, having resources sufficient to receive the software package, is provided the software package by the deployment service.

In **804**, a determination is made whether the device has sufficient resources to perform tests on the software. For example, the device may have had enough storage resources to receive the update in **802**, but may not have enough memory, processing power, or storage resources to perform tests on the received software. In such a case in which the device may not have enough resources to perform the testing, the system performing the process may proceed to **806**, whereupon the software update may be provided to a testing framework of a deployment service for testing. As noted, the testing framework may include container instances configured to simulate how the software package would be installed and perform on the device itself. Thus, the software package may be installed to execute within one of these container instances corresponding to the device on the testing framework.

Then, in **808**, tests may be performed on the software installed in the container instance on the testing framework in the same manner as it would have been tested on the device if the device had sufficient resources to perform the testing itself. Testing may include white-box testing and/or black-box testing. For example, tests may be designed to cause all program statements within the software package to be executed at least once and/or faults may be introduced into the software package to determine the efficacy of the testing and fault handling of the software package.

Additionally or alternatively, tests may be driven by one or more scripts, may be configured to simulate input from various input sources (e.g., touches on a touch screen, motion sensors, camera input, keyboard input, etc.), and may be configured to simulate error conditions. The software package may be instrumented to measure the execution performance of the software, trap errors, and/or provide trace information. In some embodiments, tests or test scripts may be dynamically generated based on usage of the device. For example, a deployment agent on the device may be configured to learn client/user usage patterns based on previous usage of the device and/or applications running on the device by the client/user. Then, when managing the testing, the deployment agent may implement a testing strategy of the software under test based at least in part on the usage patterns (e.g., by cloning or simulating actual events/inputs that occurred with previous versions of the software). As another example, the deployment agent may have recorded one or more series of events/inputs that cause an error with software currently installed in an active container and may feed a simulation of these events to the software being tested to determine whether a new software version fixes the error being exhibited by the current software version.

As still another example, as a user uses the applications of the device in the active container, the user input may also be sent (e.g., via the deployment agent) to the test container to assess how the test container responds to actual input. In some embodiments, testing may be performed on both the active container and the test container. For example, when the device is idle, the same tests run on the software running in the test container may be performed on the software running in the active container. In this way, the tests results from each container may be compared. Thus, it can be determined from the test results whether a software update performs as well or better than the software currently in use. Likewise, some implementations may support multiple test containers on the same device. For example, software provider may provide a different software package to each test container, perform testing on each, and compare the results to determine which of the different software packages performed the best, fastest, used the least amount of battery power, etc.

As an example, the device may be an electronic reader device having applications for downloading and displaying electronic books, applications for downloading and displaying news articles from various internet sources, and applications for converting text to audio for hearing impaired users. Thus, testing of such a device can include determining that its applications load within a certain amount of time and are otherwise installed correctly and determining whether the device can support receiving the volume and variety of network communications needed. Testing of applications can further include determining the resource usage and requirements of each application, measuring the amount of time each application takes to launch, and determining whether any application is reporting an error.

A deployment agent resident on the device may be responsible for provisioning and deprovisioning the software containers on the device, initiating the tests, obtaining results of the tests, and communicating the results of the tests to the deployment service. The deployment agent may also be responsible for monitoring the health and status (e.g.,

"idle," "ready to receive update," etc.) of the device, and communicating this health and status of the device to the deployment service.

However, if the device has sufficient resources (e.g., meets predetermined requirements for memory, processing capability, storage, etc.) for testing the software package on the device itself, the system performing the process **800** may proceed to **810**, whereupon the software package may be launched within a background container on the device, such as the test container **110** of FIG. **1**, the test container **310** of FIG. **3**, or the background containers **410** of FIG. **4**. In **812**, the tests may be performed on the executing software package in the background container. In some implementations, the testing may be caused to be performed by a deployment agent executing on the device. In other implementations, testing may be caused to be performed by instructions within the software package itself or by a container encapsulation engine on the device. In **814**, as tests are performed or after testing is complete, results of tests may be uploaded from the device to a testing framework of the deployment service.

Then, in **816**, whether the tests were performed on the device itself or at the testing framework, the results of the tests may be logged at the testing framework. In **818**, a determination is made whether the testing of the software package was successful. In some cases, a test failure may indicate one or more severity levels of the failure. Thus, failures of certain low-priority tests or certain failures having low severity (e.g., execution performance being below a threshold but within certain parameters may be of low severity) may not alone be determinative of unsuccessful testing, whereas certain other failures (e.g., data corruption, program crashes, memory overruns, etc.), even if only one occurrence, may be determinative of an unsuccessful test of the software package. Low severity failures and/or failures of certain low priority tests, however, may be determinative of unsuccessful testing if such failures exceed a threshold number of occurrences during testing. Therefore, it is contemplated that various methods of testing may be performed, and various ways of determining the failure or success of such tests may exist. If testing was unsuccessful, in **820**, this detail may be logged and/or other entities may be notified of the test failure, such as the software provider or the device user. In some cases, if the testing was unsuccessful, the test container may be deprovisioned.

Otherwise, if testing of the software package was successful, in **822**, the background container with the software package may be switched to be the new active container of the device, and the previously active container may be switched to be an inactive or background container. In some cases, the operations of **822** may be performed only when the device indicates that it is ready for the switch, such as after the device has been idle for an amount of time, suggesting that the user is not currently using the device.

Finally, in **824**, the previously active container may be deprovisioned from the device, such as for the purpose of freeing up allocated resources or making room for a new software package container. The dashed line of FIG. **8** is intended to illustrate that the operations of **824** may occur at a later time than the operations of **802-22**. For example, it may be advantageous to retain the previously active container for some minimum retention period (or even indefinitely), or until a second software update becomes available, as a hedge against a situation in which issues arise with the software package that were not discovered during testing. In such a situation, the deployment service may send a rollback notification to the device, and, in response, a deployment agent on the device may cause the previously active container to be quickly reactivated to be the active container on the device (rendering the new active container inactive), and the device may be thereby rolled back to its previous state before the software package was installed. Otherwise, after the retention period has passed, the previously active container may be deprovisioned as needed. Note also that one or more of the operations performed in **802-24** may be performed in various orders and combinations, including in parallel.

Note that, unless otherwise specified, use of expressions regarding executable instructions (also referred to as code, applications, agents, etc.) performing operations that instructions do not ordinarily perform unaided (e.g., transmission of data, calculations, etc.) in the context of describing disclosed embodiments denote that the instructions are being executed by a machine, thereby causing the machine to perform the specified operations.

FIG. **9** illustrates aspects of an example environment **900** for implementing aspects in accordance with various embodiments. As will be appreciated, although a web-based environment is used for purposes of explanation, different environments may be used, as appropriate, to implement various embodiments. The environment includes an electronic client device **902**, which can include any appropriate device operable to send and/or receive requests, messages or information over an appropriate network **904** and, in some embodiments, convey information back to a user of the device. Examples of such client devices include personal computers, cell phones, handheld messaging devices, laptop computers, tablet computers, set-top boxes, personal data assistants, embedded computer systems, electronic book readers, and the like. The network **904** can include any appropriate network, including an intranet, the Internet, a cellular network, a local area network, a satellite network or any other network and/or combination thereof. Components used for such a system can depend at least in part upon the type of network and/or environment selected. Protocols and components for communicating via such a network are well known and will not be discussed in detail. Communication over the network **904** can be enabled by wired or wireless connections and combinations thereof. In this example, the network **904** includes the Internet, as the environment includes a web server **906** for receiving requests and serving content in response thereto, although for other networks an alternative device serving a similar purpose could be used as would be apparent to one of ordinary skill in the art.

The illustrative environment includes an application server **908** and a data store **910**. It should be understood that there could be several application servers, layers or other elements, processes or components, which may be chained or otherwise configured, which can interact to perform tasks such as obtaining data from an appropriate data store. Servers, as used, may be implemented in various ways, such as hardware devices or virtual computer systems. In some contexts, servers may refer to a programming module being executed on a computer system. As used, unless otherwise stated or clear from context, the term "data store" refers to any device or combination of devices capable of storing, accessing and retrieving data, which may include any combination and number of data servers, databases, data storage devices and data storage media, in any standard, distributed, virtual or clustered environment. The application server **908** can include any appropriate hardware, software and firmware for integrating with the data store **910** as needed to execute aspects of one or more applications for the electronic client device **902**, handling some or all of the data

access and business logic for an application. The application server **908** may provide access control services in cooperation with the data store **910** and is able to generate content including, text, graphics, audio, video and/or other content usable to be provided to the user, which may be served to the user by the web server **906** in the form of HyperText Markup Language ("HTML"), Extensible Markup Language ("XML"), JavaScript, Cascading Style Sheets ("CSS"), or another appropriate client-side structured language. Content transferred to a client device may be processed by the electronic client device **902** to provide the content in one or more forms including, forms that are perceptible to the user audibly, visually and/or through other senses including touch, taste, and/or smell. The handling of all requests and responses, as well as the delivery of content between the electronic client device **902** and the application server **908**, can be handled by the web server **906** using PHP: Hypertext Preprocessor ("PHP"), Python, Ruby, Perl, Java, HTML, XML, or another appropriate server-side structured language in this example. It should be understood that the web server **906** and application server **906** are not required and are merely example components, as structured code discussed can be executed on any appropriate device or host machine as discussed elsewhere. Further, operations described as being performed by a single device may, unless otherwise clear from context, be performed collectively by multiple devices, which may form a distributed and/or virtual system.

The data store **910** can include several separate data tables, databases, data documents, dynamic data storage schemes and/or other data storage mechanisms and media for storing data relating to a particular aspect of the present disclosure. For example, the data store **910** may include mechanisms for storing production data **912** and user information **916**, which can be used to serve content for the production side. The data store **910** also is shown to include a mechanism for storing log data **914**, which can be used for reporting, analysis or other purposes. It should be understood that there can be many other aspects that may need to be stored in the data store **910**, such as page image information and access rights information, which can be stored in any of the above listed mechanisms as appropriate or in additional mechanisms in the data store **910**. The data store **910** is operable, through logic associated therewith, to receive instructions from the application server **908** and obtain, update or otherwise process data in response thereto. The application server **908** may provide static, dynamic or a combination of static and dynamic data in response to the received instructions. Dynamic data, such as data used in web logs (blogs), shopping applications, news services and other applications may be generated by server-side structured languages as described or may be provided by a content management system ("CMS") operating on, or under the control of, the application server **908**. In one example, a user, through a device operated by the user, might submit a search request for a certain type of item. In this case, the data store **910** might access the user information **916** to verify the identity of the user and can access the catalog detail information to obtain information about items of that type. The information then can be returned to the user, such as in a results listing on a web page that the user is able to view via a browser on the electronic client device **902**. Information for a particular item of interest can be viewed in a dedicated page or window of the browser. It should be noted, however, that embodiments of the present disclosure are not necessarily limited to the context of web pages, but may be more generally applicable to processing requests in general, where the requests are not necessarily requests for content.

Each server typically will include an operating system that provides executable program instructions for the general administration and operation of that server and typically will include a computer-readable storage medium (e.g., a hard disk, random access memory, read only memory, etc.) storing instructions that, when executed by a processor of the server, allow the server to perform its intended functions. Suitable implementations for the operating system and general functionality of the servers are known or commercially available and are readily implemented by persons having ordinary skill in the art, particularly in light of the disclosure.

The environment, in one embodiment, is a distributed and/or virtual computing environment utilizing several computer systems and components that are interconnected via communication links, using one or more computer networks or direct connections. However, it will be appreciated by those of ordinary skill in the art that such a system could operate equally well in a system having fewer or a greater number of components than are illustrated in FIG. **9**. Thus, the depiction of the example environment **900** in FIG. **9** should be taken as being illustrative in nature and not limiting to the scope of the disclosure.

The various embodiments further can be implemented in a wide variety of operating environments, which in some cases can include one or more user computers, computing devices or processing devices that can be used to operate any of a number of applications. User or client devices can include any of a number of general purpose personal computers, such as desktop, laptop or tablet computers running a standard operating system, as well as cellular, wireless and handheld devices running mobile software and capable of supporting a number of networking and messaging protocols. Such a system also can include a number of workstations running any of a variety of commercially available operating systems and other known applications for purposes such as development and database management. These devices also can include other electronic devices, such as dummy terminals, thin-clients, gaming systems and other devices capable of communicating via a network. These devices also can include virtual devices such as virtual machines, hypervisors and other virtual devices capable of communicating via a network.

Various embodiments of the present disclosure utilize a network that would be familiar to those skilled in the art for supporting communications using any of a variety of commercially-available protocols, such as Transmission Control Protocol/Internet Protocol ("TCP/IP"), User Datagram Protocol ("UDP"), protocols operating in various layers of the Open System Interconnection ("OSI") model, File Transfer Protocol ("FTP"), Universal Plug and Play ("UpnP"), Network File System ("NFS"), Common Internet File System ("CIFS") and AppleTalk. The network **904** can be, for example, a local area network, a wide-area network, a virtual private network, the Internet, an intranet, an extranet, a public switched telephone network, an infrared network, a wireless network, a satellite network, and any combination thereof.

In embodiments utilizing a web server, the web server can run any of a variety of server or mid-tier applications, including Hypertext Transfer Protocol ("HTTP") servers, FTP servers, Common Gateway Interface ("CGP") servers, data servers, Java servers, Apache servers, and business application servers. The server(s) also may be capable of executing programs or scripts in response to requests from

user devices, such as by executing one or more web applications that may be implemented as one or more scripts or programs written in any programming language, such as Java®, C, C# or C++, or any scripting language, such as Ruby, PHP, Perl, Python or TCL, as well as combinations thereof. The server(s) may also include database servers, including those commercially available from Oracle®, Microsoft®, Sybase®, and IBM® as well as open-source servers such as MySQL, Postgres, SQLite, MongoDB, and any other server capable of storing, retrieving, and accessing structured or unstructured data. Database servers may include table-based servers, document-based servers, unstructured servers, relational servers, non-relational servers or combinations of these and/or other database servers.

The environment can include a variety of data stores and other memory and storage media as discussed above. These can reside in a variety of locations, such as on a storage medium local to (and/or resident in) one or more of the computers or remote from any or all of the computers across the network 9. In a particular set of embodiments, the information may reside in a storage-area network ("SAN") familiar to those skilled in the art. Similarly, any necessary files for performing the functions attributed to the computers, servers or other network devices may be stored locally and/or remotely, as appropriate. Where a system includes computerized devices, each such device can include hardware elements that may be electrically coupled via a bus, the elements including, for example, a central processing unit ("CPU" or "processor"), an input device (e.g., a mouse, keyboard, controller, touch screen or keypad), and an output device (e.g., a display device, printer or speaker). Such a system may also include one or more storage devices, such as disk drives, optical storage devices and solid-state storage devices such as random access memory ("RAM") or read-only memory ("ROM"), as well as removable media devices, memory cards, flash cards, etc.

Such devices also can include a computer-readable storage media reader, a communications device (e.g., a modem, a wireless or wired network card, an infrared communication device, etc.), and working memory as described above. The computer-readable storage media reader can be connected with, or configured to receive, a computer-readable storage medium, representing remote, local, fixed, and/or removable storage devices as well as storage media for temporarily and/or more permanently containing, storing, transmitting, and retrieving computer-readable information. The system and various devices also typically will include a number of software applications, modules, services, or other elements located within a working memory device, including an operating system and application programs, such as a client application or web browser. It should be appreciated that alternate embodiments may have numerous variations from that described above. For example, customized hardware might also be used and/or particular elements might be implemented in hardware, software (including portable software, such as applets) or both. Further, connection to other computing devices such as network input/output devices may be employed.

Storage media and computer readable media for containing code, or portions of code, can include any appropriate media known or used in the art, including storage media and communication media, such as, volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage and/or transmission of information such as computer readable instructions, data structures, program modules or other data, including RAM, ROM, Electrically Erasable Programmable Read-Only Memory ("EEPROM"), flash memory or other memory technology, Compact Disc Read-Only Memory ("CD-ROM"), digital versatile disk (DVD) or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices or any other medium which can be used to store the desired information and which can be accessed by the system device. Based on the disclosure and teachings provided, a person of ordinary skill in the art will appreciate other ways and/or methods to implement the various embodiments.

The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense. However, it will be evident that various modifications and changes may be made thereunto without departing from the broader spirit and scope of the invention as set forth in the claims.

Other variations are within the spirit of the present disclosure. Thus, while the techniques are susceptible to various modifications and alternative constructions, certain illustrated embodiments thereof are shown in the drawings and have been described above in detail. It should be understood, however, that there is no intention to limit the invention to the specific form or forms disclosed, but on the contrary, the intention is to cover all modifications, alternative constructions and equivalents falling within the spirit and scope of the invention, as defined in the appended claims.

The use of the terms "a," "an," and "the" and similar referents in the context of describing the embodiments (especially in the context of the following claims) are to be construed to cover both the singular and the plural, unless otherwise indicated or clearly contradicted by context. The terms "comprising," "having," "including" and "containing" are to be construed as open-ended terms (i.e., meaning "including, but not limited to,") unless otherwise noted. The term "connected," when unmodified and referring to physical connections, is to be construed as partly or wholly contained within, attached to or joined together, even if there is something intervening. Recitation of ranges of values are merely intended to serve as a shorthand method of referring individually to each separate value falling within the range, unless otherwise indicated and each separate value is incorporated into the specification as if it were individually recited. The use of the term "set" (e.g., "a set of items") or "subset" unless otherwise noted or contradicted by context, is to be construed as a nonempty collection comprising one or more members. Further, unless otherwise noted or contradicted by context, the term "subset" of a corresponding set does not necessarily denote a proper subset of the corresponding set, but the subset and the corresponding set may be equal.

Conjunctive language, such as phrases of the form "at least one of A, B, and C," or "at least one of A, B and C," is understood with the context as used in general to present that an item, term, etc., may be either A or B or C, or any nonempty subset of the set of A and B and C, unless specifically stated otherwise or otherwise clearly contradicted by context. For instance, in the illustrative example of a set having three members, the conjunctive phrases "at least one of A, B, and C" and "at least one of A, B and C" refer to any of the following sets: {A}, {B}, {C}, {A, B}, {A, C}, {B, C}, {A, B, C}. Thus, such conjunctive language is not generally intended to imply that certain embodiments require at least one of A, at least one of B and at least one of C each to be present.

Operations of processes described can be performed in any suitable order unless otherwise indicated or otherwise

clearly contradicted by context. Processes described (or variations and/or combinations thereof) may be performed under the control of one or more computer systems configured with executable instructions and may be implemented as code (e.g., executable instructions, one or more computer programs or one or more applications) executing collectively on one or more processors, by hardware or combinations thereof. The code may be stored on a computer-readable storage medium, for example, in the form of a computer program comprising instructions executable by one or more processors. The computer-readable storage medium may be non-transitory.

The use of any examples, or exemplary language (e.g., "such as") provided, is intended merely to better illuminate embodiments of the invention and does not pose a limitation on the scope of the invention unless otherwise claimed. No language in the specification should be construed as indicating any non-claimed element as essential to the practice of the invention.

Embodiments of this disclosure are described, including the best mode known to the inventors for carrying out the invention. Variations of those embodiments may become apparent to those of ordinary skill in the art upon reading the foregoing description. The inventors expect skilled artisans to employ such variations as appropriate and the inventors intend for embodiments of the present disclosure to be practiced otherwise than as specifically described. Accordingly, the scope of the present disclosure includes all modifications and equivalents of the subject matter recited in the claims appended hereto as permitted by applicable law. Moreover, any combination of the above-described elements in all possible variations thereof is encompassed by the scope of the present disclosure unless otherwise indicated or otherwise clearly contradicted by context.

All references, including publications, patent applications, and patents, cited are hereby incorporated by reference to the same extent as if each reference were individually and specifically indicated to be incorporated by reference and were set forth in its entirety.

What is claimed is:

1. A computer-implemented method, comprising:

providing a portion of a software package to a client device to launch the portion of the software package in a first software container designated as background, based at least in part on a set of conditions being satisfied for the software package to be provided, while running a second version of the software package in a second software container designated as active;

obtaining, from the client device, results of a test the client device performed on the portion of the software package executing in the first software container;

as a result of the results indicating that the testing was successful, notifying the client device to make the first software container active and make the second software container run as background for at least a predetermined amount of time while the first software container is running as active; and

as a result of determining that the client device has sufficient resources to execute the software package in the second software container and prior to the expiration of the predetermined amount of time, notifying the client device to make the second software container active and the first software container run as background based at least in part on results from executing the software package in the first software container.

2. The computer-implemented method of claim **1**, wherein notifying the client device to make the first software container active and make the second software container run as background includes instructions to retain the second software container and deprovision the second software container after expiration of the predetermined amount of time.

3. The computer-implemented method of claim **2**, further comprising sending a rollback notification to the client device that the portion of the software package is to be rolled back to the second version of the software package.

4. The computer-implemented method of claim **1**, wherein the set of conditions further comprise a schedule for the software package to be provided, wherein the schedule comprises providing the software package immediately, at a predetermined date and time, and/or as a result of obtaining an indication from the client device that an idle time of the client device exceeded a threshold.

5. A system, comprising:

at least one processor; and

a memory coupled to the at least one processor, wherein the memory stores instructions, wherein the instructions are executable by the at least one processor to:

obtain, from a provider of software, a first version of a software package;

determine that a client device is capable of receiving the first version of the software package;

provide the first version of the software package to the client device, based at least in part on a set conditions, for launching as a set of instructions executing in a test container designated as background while running a second version of the software package in a second container designated as active;

obtain, from the client device, results of a test performed by the client device on the set of instructions executing in the test container;

select, based at least in part on the results, an operation to perform in connection with the test container;

cause the client device to perform the operation in connection with running the test container while making the second container run as background for a predetermined amount of time; and

in response to the results from performing the operation in connection with running the test container and determining that the client device has sufficient resources to execute the operation in the second container and prior to the expiration of the predetermined amount of time, cause the client device to perform the operation in connection with the second container while making the test container run as background.

6. The system of claim **5**, wherein the instructions are further executable by the at least one processor to provide the first version of the software package to the client device in multiple portions, each of the multiple portions provided to the client device according to a schedule.

7. The system of claim **5**, wherein:

the results of the test indicate the testing was successful; and

the operation comprises making the test container an active container.

8. The system of claim **7**, wherein the operation further comprises making the second container an inactive container.

9. The system of claim **8**, wherein the instructions are further executable by the at least one processor to notify the client device that the test container should be made an inactive container and that the second container should be made an active container when issues arise while performing the operation in connection with running the test container.

10. The system of claim 5, wherein the instructions are further executable by the at least one processor to:

query the client device to determine if the client device is ready to receive the first version of the software package; and

obtain a notification from the client device indicating the client device is ready to receive the first version of the software package.

11. The system of claim 5, wherein the instructions are further executable by the at least one processor to:

obtain configuration information from the client device; and

determine, based at least in part on the configuration information, whether the client device has sufficient resources to execute the first version of the software package in the test container.

12. The system of claim 5, wherein an operating system executing in the test container is different from an operating system executing in the second container.

13. A non transitory computer readable storage medium having stored thereon executable instructions that, as a result of execution by one or more processors of a computer system, cause the computer system to at least:

provide, in accordance with a set of conditions, a first version of a software package to client device to be launched as a set of instructions executing in a test container designated as inactive while executing a second version of the software package as a set of instructions in a second container designated as active;

determine whether launch of the first version of the software package on the client device was successful;

cause the client device to perform an operation by at least running the test container in accordance with whether launch of the first version of the software package on the client device was successful while running the second container as inactive for a predetermined amount of time; and

notify the client device to make the second container run as active and the test container run as inactive, prior to the expiration of the predetermined amount of time, as a result of execution results from the launch of the first version of the software package and a determination that the client device has sufficient resource to execute the operation in the second container.

14. The non-transitory computer-readable storage medium of claim 13, wherein the operation comprises:

notifying the client device to make the test container an active container; and

notifying the client device to make the second container an inactive container.

15. The non-transitory computer-readable storage medium of claim 13, wherein the first version of the software package corresponds to a set of libraries lacked by the client device.

16. The non-transitory computer-readable storage medium of claim 13, wherein the first version of the software package corresponds to a new version of a set of libraries that is on the client device.

17. The non-transitory computer-readable storage medium of claim 13, determining whether launch of the first version of the software package on the client device was successful comprises obtaining results of a set of tests performed by the client device on the set of instructions executing in the test container.

18. The non-transitory computer-readable storage medium of claim 17, wherein:

the results of the set of tests comprise a first set of results and a second set of results, wherein the first set of results comprises a set of results of a first set of tests executed by the client device on the set of instructions executing in the test container, and the second set of results comprises a set of results of a second set of tests executed by the client device on the second set of instructions executing in the second container; and

determining whether launch of the first version of the software package was successful comprises comparing the first set of results with the second set of results.

19. The non-transitory computer-readable storage medium of claim 13, further comprising instructions that, as a result of execution by the one or more processors of the computer system, cause the computer system to send a rollback notification to the client device that the first version of the software package is to be rolled back to the second version of the software package.

20. The non-transitory computer-readable storage medium of claim 13, further comprising instructions that, as a result of execution by the one or more processors of the computer system, cause the computer system to:

obtain configuration information from the client device; and

creating, based at least in part on the configuration information, a content of the first version of the software package.

* * * * *